FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Interactive Collaboration Platform in Augmented Reality

João Pedro Miranda Maia

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Pedro da Silva Nóbrega

Second Supervisor: João Tiago Pinheiro Neto Jacob

July 13, 2017

Interactive Collaboration Platform in Augmented Reality

João Pedro Miranda Maia

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Professor António Miguel Pontes Pimenta Monteiro
External Examiner: Professor António R. Fernandes
Supervisor: Professor Rui Pedro da Silva Nóbrega
July 13, 2017

Abstract

Numerous tools are used for remote collaboration; however, they restrict the users to a very confined virtual space and restrain interaction with the remote physical world. Augmented reality (AR) can be used to address these issues by linking the virtual and real world, thus providing new ways of interaction.

There are several AR applications that target remote collaboration. These applications consist usually in live video stream being transferred from a local to a remote user. Then, different techniques are used to augment reality using the video stream.

This dissertation proposes a collaboration framework in augmented reality based on a client-server architecture in which the remote user prepares a set of instructions in a virtual environment. Then, the client application is used to display the instructions through augmented reality. The client sends feedback back to the server and if the action was well performed, the remote user switches to the next set of instructions.

To evaluate the proposed framework, a study involving users interacting in three different experiments was performed. The participants were divided in groups of two and each group performed three experiments. One of the goals of study was to compare the developed solution with another non-AR communication software, in particular, video call.

It is expected that the proposed solution contributes to an improvement in remote collaborative work in general.

Resumo

Várias ferramentas são usadas em colaboração remota. No entanto, estas restringem os utilizadores a um espaço virtual muito confinado e limitam a interação com o mundo físico remoto. A realidade aumentada (AR) pode ser usada para abordar essas questões, permitindo uma ligação entre o mundo virtual e real, proporcionando assim novas formas de interação.

Existem algumas aplicações AR que visam a colaboração remota. Estas aplicações consistem geralmente em transmissão de vídeo ao vivo sendo transferida de um utilizador local para um remoto. Deste modo, diferentes técnicas são usadas para aumentar a realidade usando a informação do vídeo.

Esta dissertação propõe uma *framework* de colaboração em realidade aumentada, tendo por base uma arquitetura cliente-servidor, em que o utilizador remoto prepara um conjunto de instruções num ambiente virtual. De seguida, o aplicação do cliente é utilizada para mostrar as instruções através de realidade aumentada. O cliente envia *feedback* de volta para o servidor e, se a ação tiver sido bem completada, o utilizador remoto avança para o próximo conjunto de instruções.

Para avaliar a solução proposta, realizou-se um estudo envolvendo vários utilizadores que participaram em três experiências diferentes. Os participantes foram divididos em grupos de dois e cada grupo realizou as três experiências. O estudo também tinha como objectivo comparar a solução desenvolvida com outras ferramentas de comunicação que não utilizassem AR, em particular, chamada de vídeo.

Espera-se que a solução proposta contribua para melhorar o trabalho colaborativo e remoto, de uma forma geral.

Acknowledgements

In the first place, I would like to thank my supervisors, Professors Rui Nóbrega and João Jacob, for mentoring and helping me accomplish this project. I would also like to thank everyone that participated on the performed study for their time and patience. Last but not least, I would like to thank my family and friends, in particular Ana Alves, for supporting me and making me who I am.

João Pedro Miranda Maia

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

API	Application programming interface
AR	Augmented reality
BRIEF	Binary robust independent elementary features
CSCW	Computer supported cooperative work
DOF	Degrees of Freedom
FAST	Features from accelerated segment test
GPS	Global positioning system
HMD	Head-mounted display
HMPD	Head-mounted projective display
HOG	Histogram of oriented gradients
ID	Identifier
IMU	Inertial measurement unit
IP	Internet Protocol
MAR	Mobile augmented reality
MVC	Model-view-controller
NFT	Natural feature tracking
OS	Operating system
SDK	Software development kit
SIFT	Scale-invariant feature transform
SLAM	Simultaneous localisation and mapping
SURF	Speeded up robust features
UML	Unified Modeling Language
VR	Virtual reality
VRD	Virtual retina displays

Chapter 1

Introduction

Augmented reality (AR) has the potential to create new forms of interaction with technology. It provides a new view of the real world, by adding new graphic content, including 3D geometry, animations, images and video. AR is commonly used to enhance a person's perception of reality. One of the areas that augmented reality can be useful is in collaboration systems. This dissertation aims to develop a platform to improve communication and interactivity between users of these systems.

This chapter provides a brief description of the dissertation. The motivation, context, goals and contributions are also presented. Finally, it is given an overview of the structure of this document.

1.1 Motivation and Context

Collaborative software enables cooperation of people that are working on a similar task. When users do not share the same location, they can use remote collaboration tools to communicate and work together to solve a particular task. These tools include text-based applications, such as Slack, or video conferencing programs, similar to Skype. However, many times, remote collaboration involves exploring and manipulating the physical environment, and these tools are not well suited for this context. Additionally, explaining how to perform a certain task can be quite difficult, specifically, if that task involves interacting with real objects in a remote environment.

To address these issues, augmented reality can be used to enhance remote collaboration. Augmented reality refers to the superposition of virtual information over the real world, that is added to what the user perceives naturally, enhancing one's current perception of reality [RJA+16; GZB13; ZD03]. Augmented reality applications are often deployed on smartphones or on special equipment including head-mounted displays. In addition, AR systems need to accurately align virtual objects on real images, otherwise the objects may undesirably flicker. A common solution to this

issue is to place fiducial markers on the physical world that are easily identified by the AR systems and, thus, provide the information required to display stable virtual content (e.g. AR Toolkit[1]).

In the case of guiding in remote tasks, augmented reality can be used to transmit ideas in different manners by allowing remote users to create virtual content in a local user's environment. This is a very welcomed feature for deaf people, as they can visually see in-situ how tasks can be performed, without having to rely on other resources such as video. In similar contexts, augmented reality has also been used as a guide for maintenance and assembly tasks [HF09; KM13; ZBW+14], as these operations are often very complex, and for learning purposes such as learning descriptive geometry [RJA+16], and teaching electrical circuits concepts [RCR+14].

Remote collaboration presents several challenges in terms of interaction. On one hand, augmented reality can enhance the communication from the point of view of the local user, who pretends to perform a certain task. On the other hand, the remote user needs to interact with the collaboration tool in a very different way in order to create the virtual instructions. In addition, augmented reality prompts problems by itself. AR applications often require the use of fiducial markers, which, in practice, may require the use of prepared working environments [RCR+14], or be undesirable. In alternative, markerless AR systems do not require fiducial markers, yet, they are often not very robust, or they require a previous computation of the visual features of the scene [ZBW+14]. Furthermore, augmented reality requires the use of common hand-held devices, such as smartphones, or the use of other special equipment, for instance head-worn displays, for hands-free experiences.

1.2 Research Questions

Remote communication between users can be ambiguous and hard to express. Human beings communicate with each other not only by words but also by other non-verbal ways. Body language is one visual cue that is very important to communication as it conveys important information. Many of these non-verbal cues are lost in remote communication which can difficult the transmission of ideas and thoughts, especially when these refer to a remote environment. In addition, many times communication can be eased if both users can see or share the same environment. Remote collaboration often involves solving complex tasks over a particular remote environment and, in these conditions, transmitting complicated ideas and concepts can be a problem.

The purpose of this dissertation is to explore how augmented reality can improve the transmission of ideas and lower the barriers of communication in collaborative remote systems. Hence, this dissertation aims to answer the following research questions:

- Does augmented reality improve teaching of complex tasks over a particular object in remote collaborative systems?

[1] Open source augmented reality sdk | artoolkit.org. URL: `https://archive.artoolkit.org/` (visited on 06/26/2017).

This is the main research question that this dissertation aims to respond. With the goal to answer it, a prototype AR application was developed based on a framework designed to enhance remote collaboration, which was tested and evaluated.

- Are collaborative AR systems preferred by collaboration work groups over typical communication tools?

  Collaborative AR systems should be no more complex than typical and widely-used communication tools. Failing to accomplish this issue, collaborative AR systems might be not well accepted by the population that already use certain tools. These systems should provide enough distinctive and useful features so that users are willing to prefer them over other tools.

- Are collaborative AR interfaces more intuitive to use than typical collaboration tools?

  Collaborative AR interfaces should also be intuitive and easy to use so that they provide no interaction barriers to new users. These AR systems should support natural ways of interaction in order to support a wide range of different users, actions and environments.

It is expected that these questions are answered by an evaluation of the solution that is described throughout this dissertation.

1.3 Objectives

To answer the research questions presented in the Section 1.2, the following objectives were proposed:

- research state-of-art AR collaborative systems and investigate AR-supporting technologies;
- develop a framework that enables remote collaboration in augmented reality, in which a user creates instructions in a virtual environment;
- design an architecture that supports the previously described framework;
- implement a functional platform following the architecture and mentioned framework;
- test and evaluate the framework, using the developed platform;
- investigate how remote communication can be improved in order to enhance and coordinate teaching of complex tasks.

Researching related work regarding collaborative systems and investigating AR technologies and user interfaces related with the subject provided a solid basis for developing the AR collaboration framework. The developed platformed put in practice the framework and allowed its testing and validation.

1.4 Proposed Solution

To address the issues mentioned in Section 1.1, this dissertation proposes a framework for remote collaboration in augmented reality. The framework allows a mentor to explain a complex task, by dividing it in several simple steps and by describing each step using a set of instructions. These instructions can be anything that provides helpful information for the corresponding step, such as images, 3D objects, text or even visual animations. The information is saved and can be used for future collaboration sessions in order to guide an apprentice perform the task.

For a particular task, the virtual environment used by the mentor to create the instructions must have represented, in some way, the objects that the apprentice may need to use to complete the task in the real world. For instance, for the mentor to be able to teach how to change a bicycle's tire, the bicycle must be represented in order to create the right instructions in their correct place. However, these objects must not appear to the apprentice in form of augmented reality, as the objects themselves are already represented in their physical form. Thus, these objects need to be represented in the same scale as the real or some kind of mapping between the virtual and the real world must be made, otherwise the instructions will not be appear on the correct places using augmented reality.

During the collaboration sessions, the mentor explains the task to the apprentice, step by step, by presenting the corresponding instructions through augmented reality. During each step, the mentor needs to validate the actions perform by the apprentice, in order to clarify any doubts the apprentice has and to advance to the following step. This validation depends on the implementation and may be adapted for several use cases. For instance, a video stream transmitted by the apprentice's system could be useful; however, in cases in which network bandwidth is a problem, simpler solutions could be used, such as transferring a photo of the step solution.

In order to evaluate the solution, an implementation based on the framework, named CooP AR (Collaboration Platform in Augmented Reality) was developed and is presented, along the framework, in Chapter 3.

1.5 Contributions

One of the contributions of this dissertation is providing an overview of related work regarding collaboration systems, augmented reality and interaction. AR-supporting technologies are also presented, including an overview of feature detection and description, visual tracking techniques, software solutions and AR-enabling hardware.

One of the main contributions of this dissertation is the elaboration and description of an AR collaboration platform that enables remote cooperation to solve complex tasks, that is flexible to allow a wide range of scenarios that include low network bandwidth and user disabilities (e.g. hearing impairment).

This dissertation also contributes with an implementation of the proposed framework. The implementation is composed by two application that communicate through the network. One of

the applications enables the creation of instructions and mentoring of tasks, whereas the other provides the visualization of the instructions using augmented reality.

Another major contribution is having performed a study, using the developed implementation of the framework. The study had the collaboration of thirty participants in three different experiments. The efficiency of the platform on helping users execute complex tasks and the usability of the system were evaluated.

A technical paper giving an overview of the framework and the conclusions of the conducted study is being developed and will be published in the future.

1.6 Document Structure

This document is divided into four chapters, excluding the present one. In Chapter 2, a review of the literature related to augmented reality systems is performed, including presentation of existing applications, an overview of computer vision AR supporting techniques and user interfaces, and analysis of several technologies. Chapter 3 describes a collaboration framework, a software of implementation of that framework and the corresponding system architecture. Chapter 4 describes a study performed by users that used the software applications developed and presents and discusses the results of the experiments. Chapter 5 presents a summary of this project, the conclusions and future work.

Chapter 2

Literature Review

In this chapter, augmented reality related works are presented in order to show the state-of-the-art solutions and unsolved problems. There is also an overview of computer vision algorithms typically used by augmented reality systems, as well of the different ways of interacting in augmented reality applications. Finally, several potentially useful technologies and tools in the context of this project are analysed.

2.1 Augmented Reality and Interaction

Augmented reality (AR) refers to the superposition of virtual information over the real world, that is added to what the user perceives naturally, enhancing one's current perception of reality [ZD03; RJA+16; GZB13]. From a technological point of view, AR applications require real time interaction and accurate 3D registration of virtual and real objects [ZD03; RJA+16].

According to Azuma [Azu97] and Azuma et al. [ABB+01], augmented reality refers to any system that:

- combines real and virtual objects;
- aligns real and virtual objects in three dimensions, a process named *registration*;
- is interactive in real-time.

This definition is very interesting for two reasons:

- it doesn't limit AR to a specific technology, such as Head Mounted Displays (HMD), allowing other technologies to be used while retaining the essential aspects of AR;
- it doesn't limit AR to sight, as it may apply to other senses such as hearing, touch and smell [VP07].

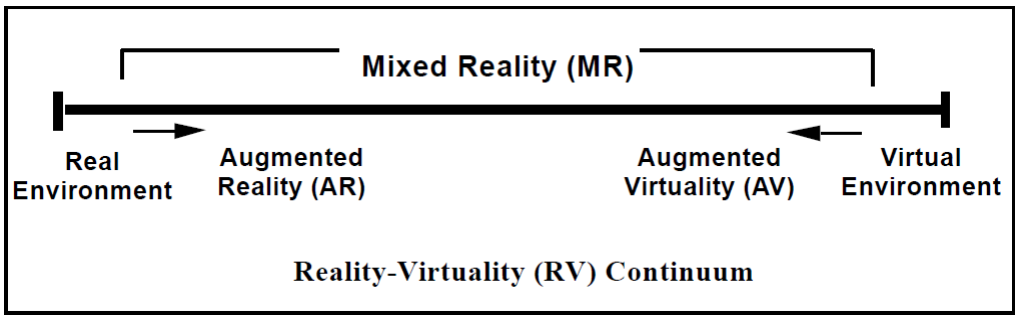


Figure 2.1: Representation of the Reality-Virtuality Continuum. Image source: [MTU+95].

Milgram et al. [MTU+95] defined a reality-virtuality continuum which relates augmented reality to other technologies such as virtual reality (VR) (see Figure 2.1). On the left extreme lies the real environment, that is, an environment which consists entirely of real objects. On the right extreme is virtual reality which is constituted solely of virtual objects. Anywhere between these two extremes is defined as mixed reality. Augmented reality lies somewhere between the left extreme and the middle of the continuum.

Augmented reality has been used in several real-case scenarios. For instance, it has been used to improve learning in classroom; to train and help mechanics in maintenance tasks; and to make communication more clear in remote collaboration systems.

2.1.1 Mobile Augmented Reality

Mobile augmented reality (MAR) applies augmented reality in truly mobile settings, without constraining the individual's whereabouts to a restricted area [RJA+16; HF04]. Smartphones are powerful enough to support this technology, and, as they are widely used, they particularly interesting for education purposes [RJA+16; WS09].

In fact, several researches point out that mobile augmented reality systems (MARS), as is the case of smartphones, enhance the learning process [AGG+15] and increase the interest and attention of the students [CHD+11; RSN07]. Educational content has a vast potential for future use in mobile devices, as they work virtually anywhere [JBC08; AGG+15].

This document presents several works that developed mobile augmented reality systems, such as Juan et al. [JBC08] and Ravé et al. [RJA+16] described in Section 2.1.4. In addition, Kourouthanassis et al. [KBL15] proposed a set of interaction design principles for the development of mobile augmented reality systems, which are described in Section 2.1.5.5.

2.1.2 Augmented Reality in Collaborative Systems

Collaborative software enables cooperation of people that are working on a similar task. When users do not share the same place, they can use remote collaboration tools to communicate and work together to solve a particular task. However, explaining how to perform a certain task can be quite difficult, specifically, if that task involves interacting with real objects in a remote environment. Augmented reality can improve this process by creating new ways of interaction and

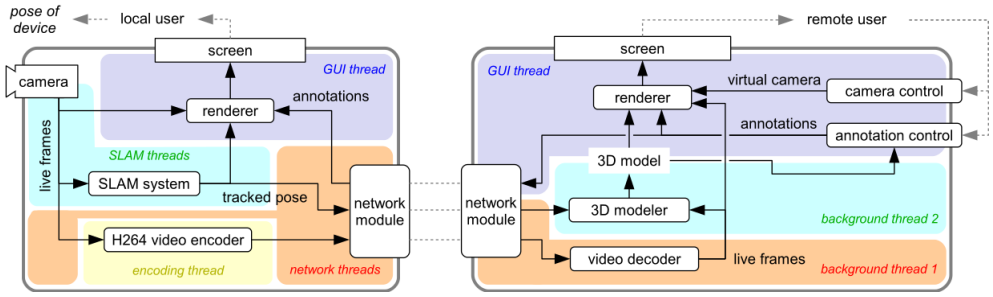


Figure 2.2: System architecture of both the local and remote users' systems. Image source: [GNT+14b].

transmission of ideas by allowing remote users to create virtual content in a local user's environment.

Gauglitz et al. [GNT+14b] present an application that supports an augmented shared visual space for live mobile remote collaboration on a particular physical task. The remote user can control a virtual camera and explore the scene independently of the local user's current camera. In addition, the remote user can communicate via spatial annotations that are immediately visible to the local user in augmented reality. The physical scene is tracked and modeled in real time and in 3D, using monocular vision-based simultaneous localization and mapping (SLAM) and subsequent surface modeling. The resultant model then supports anchoring of annotations and virtual navigation.

The local user's system runs a SLAM system and sends the tracked camera pose as well as a video stream to the remote system. It also receives information about annotations from the remote system and presents it to the local user through augmented reality.

The remote system consists of five main modules — network module, 3D modeler, camera control, annotation control, renderer — and the framework to hold them together (see Figure 2.2). The network model receives the incoming video stream and camera pose from the local user's system and then the 3D modeler component constructs a 3D surface model on the fly. The camera control module continually stores new video frames with their associated camera poses from the live video stream and the remote user can then navigate through these video frames. The camera control module supports operations such as panning, zooming and change of viewpoint, resembling tools such as Google Street View. Finally, the renderer uses the information computed by all the other modules such as the 3D model, the saved video frames, the local user's live camera video frame, the virtual camera pose and the annotations to render a final image that is presented to the remote user. In addition, the renderer uses an image-based render approach, so that as the virtual camera approaches the physical camera, the views become identical, providing a very smooth transition between the "model view" and the "live view". The remote user's interface can be seen in Figure 2.3.

However, the system presents some limitations such as assuming that the scene is static, requiring a stereo scene initialization (a general problem of SLAM systems) and the non-occlusion of annotations in the local user's system.

Figure 2.3: Remote user's interface. The local user's current view can be seen in the bottom left window. On the top right, a blue gradient indicates that it's not possible to pan beyond the extent of the model. Image source: [GNT+14b].

2.1.3 Augmented Reality for Maintenance and Assembly Tasks

AR for maintenance or assembly tasks has been proposed by several applications [HF09; KM13; ZBW+14]. As the complexity of maintenance and assembly procedures can be huge, the training of professionals to perform these tasks has been the focus of several research groups [WBE+11].

The use of an optimal AR system (no lag, high quality see-through calibration, devices integrated into work clothes, optimized interaction, weight-reduced HMD) decreases the overall strain, compared to common work assistance systems [TDM+08].

Webel et al. [WBE+11] analyses the use of augmented reality for training purposes. They state that a big advantage of AR for training is that the trainee can interact with the real world objects and simultaneously access the virtual information for guidance. This way, the trainee learns to perform the tasks by observing the augmented instructions, without the need of any physical training material (e.g. a user manual). Another advantage of using a training platform that uses virtual elements is that it allows for the measurement and evaluation of the trainees performance in ways that are not possible without using AR. In addition, it is also possible to respond to the trainee's actions and present corresponding feedback. Furthermore, it also gives the flexibility to adapt and rearrange the sub-tasks according to some performance factor, which is not possible in the real world.

However, as pointed out by Webel et al. [WBE+11], in the context of training applications, a potential danger is that users may not be able to perform the tasks without the AR instructions or when the technology fails. Therefore, the training programs should include phases in which the

amount of AR features is reduced (e.g. less virtual components and instructions), in order to teach the user and not only guide him through the task. To summarize, AR based training applications should clearly differ from AR based guiding applications.

Henderson and Feiner [HF09] implemented a prototype AR application to support military mechanics conducting routine maintenance tasks inside an armored vehicle turret (Figure 2.4). The system was composed of a head-worn display which augmented a mechanic's natural view with text, labels, arrows and animated sequences designed to facilitate the execution of the tasks.

A maintenance task can be decomposed into several sub-tasks. When the user is performing a certain maintenance sub-task (e.g., toggling a switch), the application assumes a specific state which represents that particular sub-task. For each sub-task the application provides five forms of augmented reality content to assist the mechanic:

- arrows to redirect the attention of the user to a specific component (e.g. when the component is not in line of sight);
- text instructions that provide a description of the task, as well as notes and warnings;
- labels that show the location of the target component and surrounding context;
- a close-up view depicting a 3D virtual scene on the target component at close range;
- 3D models of tools and components at target locations, to assist in more ambiguous or complex tasks.

Users use a wireless wrist-worn controller as the primary mean of interaction with the system. The controller interface is written using the Android SDK and it can be used to control the speed of animations or to replay them and to navigate between maintenance tasks. The system proved to be more effective than the current techniques used by the US Marine Corps mechanics, reducing time and head movement during the maintenance tasks.

Zhu et al. [ZBW+14] developed an AR mentoring system (AR-Mentor) to assist in maintenance and repair tasks of complex machinery, such as vehicles and industrial machinery (see Figure 2.5). In this system, the user wears a helmet-based sensor head package that includes one pair of stereo cameras, one *inertial measurement unit* (IMU) and one *head-mounted monocular display* (monocular HMD). The IMU and the stereo cameras allow a pose estimation module to calculate the 3D location and rotation of the user's head, with respect to the target equipment, with high precision. This pose estimation module works as follows:

1. A database of visual features of the training equipment was previously developed. Then, in real-time, a landmark matching block establishes the position of the user with respect to the equipment.
2. A visual and navigation module tracks visual features over time to know how the user's head is moving in real-time. This information combined with the position provided by the landmark match block allow a precise estimation of the 3D head position and orientation.

Figure 2.4: On the left image, a mechanic wears an HMD performs a maintenance task inside a military vehicle. On the right image, the augmented reality that the mechanic can see: text instructions are displayed at the top; a label identifying the target component; 3D objects are visually display in order to assist the user performing the task; a close-up view of a 3D model of the target component. Image source: [HF09].

3. Finally, a low-latency module uses the information calculated from the previous steps to predict where would the user look at the exact time the information is displayed.

This low-latency user location information is then forwarded to a rendering subsystem that generates animations that match exactly the users perspective view as overlays in the HMD.

The landmark database is created using a set of videos collected using stereo cameras. The building of the landmark database is quite complex, as the equipment usually consists of several removable and articulated parts. To address this issue, the video sequences must include all possible scenarios with moving parts. Then, each individual landmark is characterized by a *locale ID* and a *object state ID*. However, only landmarks of the equipment are useful for matching and landmarks of the surroundings degenerate matching performance. Therefore, landmarks of the surroundings have to be removed from the database; a process that is done manually. The image matching algorithm uses Histograms of Oriented Gradients (HOG) feature descriptors, commonly used for object detection. The final result of the 3D sparse point cloud can be seen in Figure 2.6.

The accuracy of the pose estimates alone are not sufficient for delivering an acceptable user experience, since, as the users head moves, the markers appear to bounce around in the display. This happens because the user has no-delay in visual perception of the real world and the render of the associated markers is too slow. In order to compensate all the latencies in the system, a low latency prediction module estimates the camera pose corresponding to a certain timestamp into the future, using the Kalman Filter [Kle96] given the orientations of past camera poses.

The AR-Mentor also features a virtual personal assistant that allows the user to interact with it, by asking questions and get specific guidance for the task at hand. A prototype application was built in order to instruct a novice to perform a 33-step maintenance task on a vehicle and the tests demonstrated that the system is able to help the users completing the tasks, allowing the instructor to cover more students and focus on higher-order teaching.

Figure 2.5: AR-Mentor System. In this case, the task consists in extracting screws to remove a shield from a military vehicle. Image source: [ZBW+14].

AR-Mentor is a complex AR application that has many features that can be applied to a remote collaboration platform. In this case, all interactions are done with a virtual assistant that guides the user in all cases, answering asked questions and displaying virtual content useful for the task at hand. These concepts could be transferred to a collaborative application in which, instead of a virtual assistant, a remote user guided all the process by supervising the process and generating virtual instructions to guide the local user.

2.1.4 Augmented Reality for Education

AR has been successfully used for many learning purposes in classroom. AR is extremely useful for learning because it can add convenient information over real objects and it allows highly interactive visual ways of learning [JBC08; RJA+16].

Juan et al. [JBC08] developed an AR system for learning the interior of the human body. The users are able to see inside the human body, by "opening" the abdomen of a virtual human body, using their own hands. A study was carried out with children of the Summer School of the Technical University of Valencia, in order to:

- understand the degree of involvement of the children;
- the perceived usefulness of the system;
- the usability of the application and interface;
- if both visualization systems (HMD or monitor) were perceived in a similar way

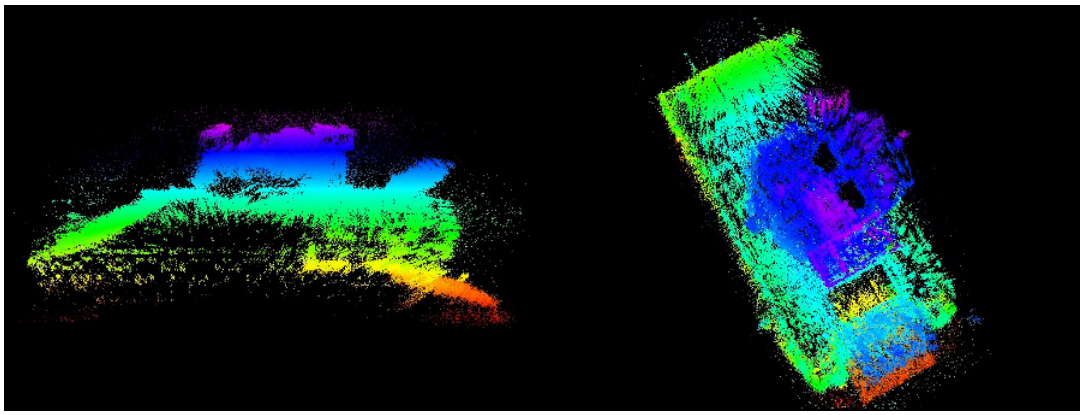


Figure 2.6: The 3D reconstructed sparse point cloud of a military vehicle used for AR-Mentor. Image source: [ZBW+14].

- if the order of exposure influenced the children's perception

The results found out that:

- children enjoyed learning with the system and consider it as useful tool not only for learning the interior of the human body but also for learning other subjects;
- both visualization modes (HMD and monitor) had similar influence in children;
- the order of exposure didn't affect their perception of the useful aspect of the system.

Ravé et al. [RJA+16] developed a mobile AR system, DiedricAR, aimed at learning descriptive geometry, that works together with a workbook that contains AR markers (see Figure 2.7). The exercises are solved step by step and each stage can be selected through virtual buttons.

It was developed using the Vuforia SDK and the development framework chosen was Unity3D.

DiedricAR was built taking into account the design guidelines proposed in [KBL15], which are also presented in Section 2.1.5.5. The authors later concluded that the design guidelines applied had a very positive impact on the user experience, as the exercises tests performed by students were very good.

2.1.5 User Interfaces for Augmented Reality

One of the most important elements in every application is the usability and interaction with the system and AR applications are no exception in this matter. Furthermore, augmented reality allows new ways of interaction that are not possible with other technologies. In this section, several modes of interaction and design guidelines for AR mobile systems are presented.

There are four main types of interaction in AR applications: tangible AR interfaces, collaborative AR interfaces, hybrid AR interfaces and multimodal interfaces.

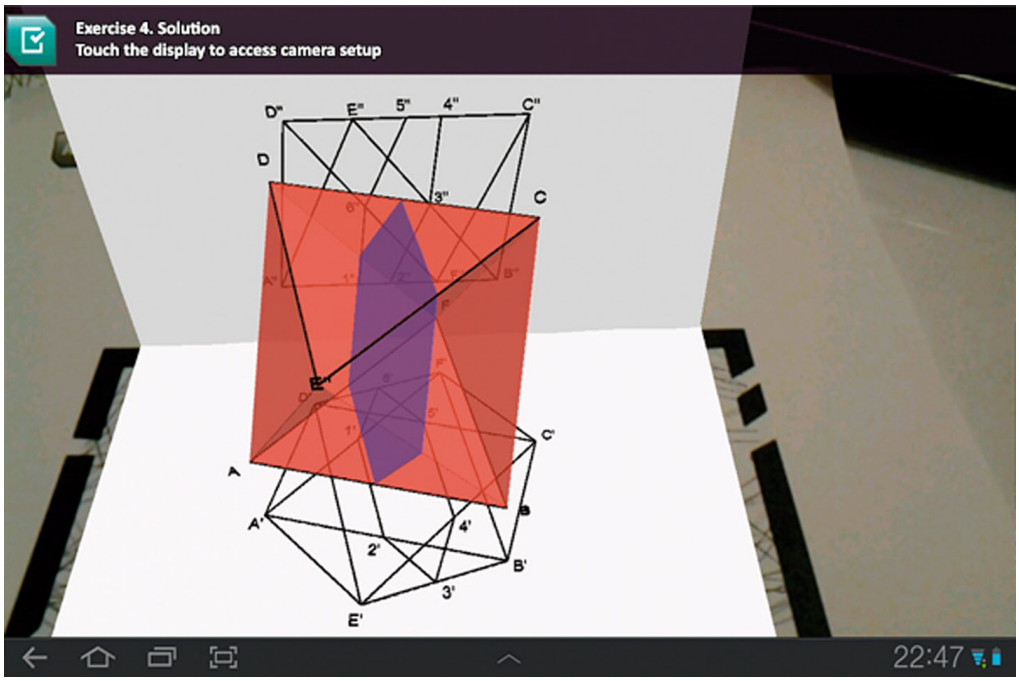


Figure 2.7: The solution for a descriptive geometry exercise displayed by DiedricAR. Image source: [RJA+16].

2.1.5.1 Tangible AR Interfaces

Tangible AR interfaces allow the user to interact with a virtual object by manipulating a corresponding physical object. That is, in tangible AR each virtual object is assigned to a physical object and virtual objects can be interacted by manipulating the corresponding tangible objects [BKP08].

Billinghurst et al. [BKP08] developed a set of design principles that an AR interface should follow in order to be intuitive to use and provide a seamless interaction with the virtual content:

- the use of physical objects for manipulating virtual content;
- support for 3D interaction, allowing the user to move, rotate and approximate the virtual objects;
- support for both time-multiplexed [1] and space-multiplexed [2] interaction;
- support interaction with multiple hands;
- match the physical constraints of the object to the requirements of interaction task;
- support for manipulation of multiple objects at the same time;
- collaboration between multiple participants.

[1] A single physical device can perform different actions in distinct points in time.
[2] Each action can be performed by a single physical device occupying its own space.

Figure 2.8: An example of tangible augmented reality. The user's writing and drawings convey key information that is sent to the software. Image source: [VLL+14].

.

An example of a tangile AR application is Strip'TIC developed by Vinot et al. [VLL+14]. Many air traffic controllers around the world use paper flight strips, which have information about the planned route of an aircraft. They write on these strips and move them on a stripboard in order to organize their mental picture of the traffic. However, the use of paper limits the automation of air traffic control (ATC) systems. With the introduction of the Strip'TIC system, paper strips are marked with Anoto Digital Pen patterns. By using a digital pen to perform their actions, all the actions are sent to the software (see Figure 2.8). The resulting strokes of the pen are both physical and digital. In addition, they can use the pen to point at symbols on the radar display or names on the paper strips. Strip'TIC provides a link between the physical and the digital worlds, while allowing controllers to manage traffic with their current rules and practices.

2.1.5.2 Collaborative AR Interfaces

Collaborative AR interfaces are used for both remote and co-located AR collaboration [CFA+11]. That is, they are useful for two use-case scenarios:

- in face-to-face collaboration, to enhance the shared physical world and create an interface for *computer supported cooperative work* (CSCW) [BK02];
- in remote collaboration, to enhance the communication between users by means of gaze and non-verbal communication cues, and manipulation of virtual objects [BK02; CFA+11].

According to Szalavári et al. [SSF+98], the key properties that characterize a co-located AR collaborative environment are:

- virtuality: the manipulation of virtual objects that don't exist in the physical world;
- augmentation: real-objects can be better understood by means of virtual descriptions or guides;
- multi-user support: multiple users can interact and collaborate with each other
- independence: each user is free to move independently of other users and define their own viewpoint;
- sharing vs indiduality: users can see the same coherent model, but can also see different data from each other, as required by the application's needs and individual's choices.

In the case of remote collaboration, AR has been used to enhance teleconferences [BFS04] and communication between users by allowing them to create annotations and navigate virtually [GNT+14b].

Gauglitz et al. [GNT+14a] present a new interface for an application previously developed by Gauglitz et al. [GNT+14b], which was already described in Section 2.1.2.

The remote user uses a touchscreen interface for all interactions with the system. The authors noted that, in several 3D applications' interfaces, absolute spatial operations (e.g. pointing to an object) are indirect and require the user to first locate a mouse cursor and position it with respect to the object of interest. A touchscreen interface has the advantage that it supports direct interaction without need of an intermediate representation such as a mouse cursor and provides haptic feedback during the touch. In addition both relative spatial operations (e.g., indicating a direction in 3D) and absolute spatial operations are very natural and intuitive.

A side pane contains several virtual buttons useful in the context of the application. A two-finger tap while the main view is coupled to the local user's live view freezes the current viewpoint. Using a single finger, the remote user can draw annotations into the scene. When the user starts to draw an annotation during the local user's live view, the viewpoint is also temporarily frozen in order to enable accurate drawing, transitioning back again to the live view as soon as the finger is lifted. Also, all navigation-based controls are triggered by two fingers.

The touchscreen interface has the inconvenient that the depth of the drawing along the current viewpoint's optical axis is unspecified. The application could require the user to specify the depth of the drawings manually; however, that wouldn't be user friendly. Having that in mind, the authors developed some techniques to infer depth automatically. The initial assumption is that the annotation is in contact with a 3D surface of the scene; that is, annotations floating in midair are not supported. Three different ways of inferring the depth were considered:

- spray paint: the annotation is created directly on the 3D surface as if spray painted onto it;

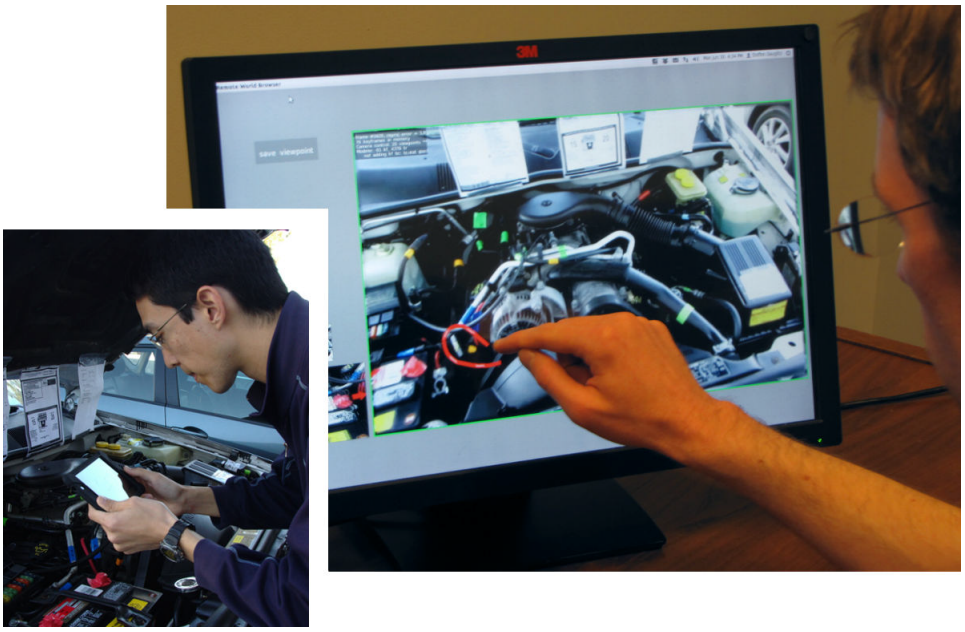


Figure 2.9: A remote user points out an element in the environment by drawing an outline around it. The annotation is displayed in three-dimensions to the local user, who is holding a tablet. Image source: [GNT+14a].

- plane orthogonal to viewing direction: the annotation is created on a plane orthogonal to the viewing direction;
- dominant surface plane: estimate a dominant plane using, for instance, RANSAC or Least Median of Squares.

A study found out that users tended to prefer the dominant surface plane method. However, neither of the depth interpretations are suited for all cases and therefore, the interface could offer to change the type of depth inference used. A limitation of the planar projection methods is that drawings are treated as different annotations as soon as the finger is lifted, which causes them to be moved to different planes. This causes unexpected behaviour to the users when they try to draw, for instance, an arrow.

2.1.5.3 Hybrid AR Interfaces

In hybrid AR interfaces, information can be displayed across various complementary displays [SOB+05]. Users can also interact through a wide range of interaction devices. These kind of interfaces are suited for situations where the exact devices and displays used by the end-user are not known in advance. Hybrid user interfaces should be built upon a flexible infrastructure that allows the use of several input devices and interaction techniques.

Sandor et al. [SOB+05] created a prototype mixed reality system that allows users to reconfigure a running hybrid user interface. The AR overlay is presented on a head-tracked, see-through, head-worn display and the system allows the user to configure input devices to perform simple 3D

transformations on virtual objects. The input devices used included game controllers, sliders and bend sensors. In order to support interactive end-user reconfiguration of the mapping between input devices, objects and operations, the system provides visual feedback that indicates the current configuration and additional visual and audio feedback when the system is reconfigured.

2.1.5.4 Multimodal AR Interfaces

Multimodal interfaces aim to support the recognition of naturally occurring forms of human language and behavior such as speech, touch, natural hand gestures or gaze and can offer interaction alternatives to better meet the needs of diverse users with a range of usage patterns and preferences [CFA+11; Tur14]. Some potential advantages of multimodal interfaces include [Tur14; OCW+00]:

- flexible use of input modes;
- they give users alternatives in their interaction techniques;
- they best suit a wider range of users, tasks, and environmental situations;
- they accommodate individual differences, such as permanent or temporary handicaps;
- they help prevent overuse of any individual mode during extended computer usage.

Some modality examples relevant to multimodal human-computer interaction include [Tur14; BG96]:

- visual: face location, gaze, facial expression, lipreading, head, hands and body gestures and sign language;
- auditory: speech and non-speech audio;
- touch: pressure, location and selection and gesture.

2.1.5.5 Mobile Augmented Reality Design Principles

Kourouthanassis et al. [KBL15] proposed a set of interaction design principles for the development of mobile augmented reality applications. The principles proposed are the following:

1. Use the context for providing content. By using the contextual information provided by sensor and marker technologies, the interface may have different behavior according to the user's task. For instance, the application may use reference markers to identify object properties or provide interactive focus and context visualization in particular situations.
2. Deliver relevant-to-the-task content. Users should not be distracted by unwanted content and, as such, the interface should only include relevant information to users. The application may use filtering or personalization techniques in order to reduce aspects of cognitive overheads.

3. Inform about content privacy. Users recognize what information about their activity will be saved, where it will be saved and who has access to the content they share. Users should be in control of their privacy and the system should allow them to change and manipulate their personal information based on their preferences. Failing to address this issue is likely to generate anxiety and deteriorate the overall user experience.

4. Provide feedback about the infrastructure's behavior. Mobile hardware specifications vary and the performance of the mobile system can affect interaction. Thus, it is essential that interfaces reflect the state of the infrastructure, such as real-time feedback regarding the system state. In addition, an application should have different configurations, each having different resource requirements, in order to achieve dynamic deployment and quality adaptation based on the infrastructure changes.

5. Support procedural and semantic memory. Familiar icons and widely-used interface metaphors should be used to communicate system functionality.

2.1.6 Summary

Augmented reality is useful in several contexts, including assisting in maintenance and assembly tasks, improving learning activities and enhancing both local and remote collaboration. Applications that use augmented reality can be very complex, involving complicated visual tracking systems and 3D reconstruction in real-time, if the scene is not known in advance. In the case of collaborative AR applications, the remote user interface usually accommodates a live video stream provided by the local user system.

Augmented reality interfaces can be classified as tangible, collaborative, hybrid or multimodal. Tangible interfaces allow the user to interact with virtual objects by manipulating a corresponding physical objects. Collaborative interfaces are used for both face-to-face and remote collaboration and aim to enhance the communication between users. The main goal of hybrid interfaces is to allow the use of multiple displays and different interaction devices depending on the daily user's needs and real-world context. Multimodal interfaces are usually preferred by users, as they aim to support the recognition of natural human language and behavior. Finally, design principles for mobile AR systems were also presented.

2.2 Computer Vision

Computer vision refers to the process of transforming data from images into either a decision or a new representation [BK08]. Computer vision techniques are on the basis of all augmented reality systems. Hence, it is important to know which techniques exist and the advantages and drawbacks of each one of them. Having some knowledge of this subject can be valuable to solve augmented reality related problems and better understand the AR systems in general. In addition, it can also be very useful for deciding which technologies are most suited to the project.

In this section, algorithms for feature detection, feature description and visual tracking are presented and discussed.

2.2.1 Feature Detection and Description

Feature detection and feature description algorithms are on the basis of many computer vision applications, such as visual tracking systems which are discussed in section 2.2.2.

A feature detection algorithm analyses an image and finds interest points that are likely to be useful for further stages of an application. These feature detector algorithms should be highly repeatable and the resulting features can correspond to edges, corners or blobs [PPS13]. These keypoints should meet some requirements and desirable properties, such as [Lin15]:

1. having a clear definition;
2. well-defined position in image space;
3. having local structures around the interest point that are rich in information content that can be useful for posterior analyses;
4. being invariant to perspective image deformations and illumination variations, so that the same interest points can be reliably detected multiple times across different images;
5. being sufficiently distinct, such that physically different points are detected as distinct interest points;
6. being scale-invariant to make it possible to match corresponding image patches under scale variations.

A feature description algorithm uses the neighborhood region around each detected feature to compute a distinctive feature description.

Scale-invariant feature transform (SIFT) [Low04] is an algorithm that detects and describes local features in images. SIFT-based feature descriptors have 128 dimensions and have the property of being invariant to scale, rotation, illumination and camera viewpoint [QSS14]. One disadvantage is that SIFT has a high computational complexity, which makes it not very suitable for real-time applications. Having this in mind, Qasaimeh et al. [QSS14] proposed an hardware architecture specifically design to accelerate the SIFT algorithm. The system was able to detect the SIFT features of VGA images in real time in around 2.2755 milliseconds and generate up to 2316 descriptors within 33 milliseconds. Khan et al. [KMW11] proposed a SIFT algorithm version that computed descriptors having only 64 dimensions. The authors proposed that it should be preferred in future applications as it is as accurate as the original version, while offering almost three times faster image matching and half the memory requirements.

Speeded up robust features (SURF) [BTV06] is a faster alternative to the SIFT algorithm. Khan et al. [KMW11] performed a study to evaluate SIFT and SURF against various image deformations on a benchmark dataset. They concluded that SURF is generally as good as SIFT except when testing scaling, large blur and viewpoint invariance.

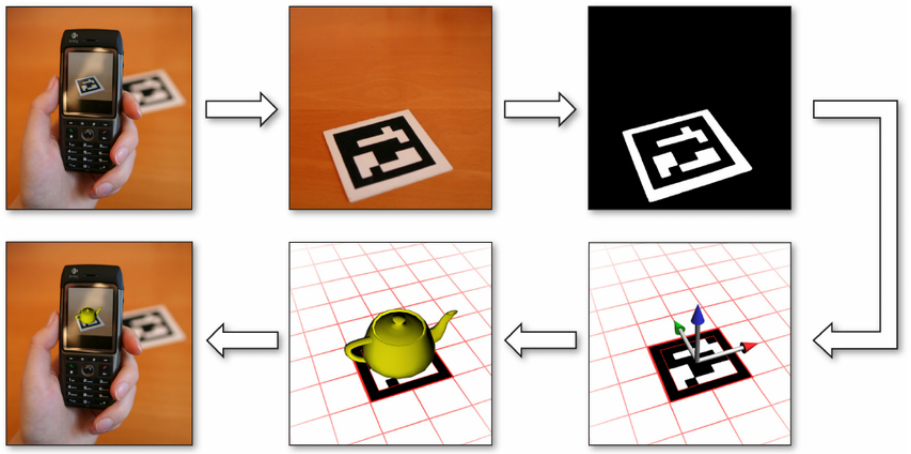


Figure 2.10: General workflow of an AR application using fiducial markers. Image source: [WS07].

Oriented FAST and Rotated BRIEF (ORB) [RRK+11] is an alternative algorithm to SIFT and SURF. It is built on a feature detector called *features from accelerated segment test* (FAST) [RD06] and a feature descriptor named *binary robust independent elementary features* (BRIEF) [CLS+10], as both techniques have good performance and low computational cost. To validate ORB, the authors performed experiments that tested the properties of ORB relative to SIFT and SURF. The results demonstrated that ORB had good performance and efficiency relative to SIFT and SURF, being an order of magnitude faster than SURF and over two orders faster than SIFT.

2.2.2 Visual Tracking

Visual tracking refers to the process of tracking objects by analyzing non-stationary image streams that change over time [RLL+08]. Applications that use visual tracking include augmented reality, robotics, surveillance and vehicle tracking [ML09].

In particular, augmented reality applications need to accurately align virtual objects on real images, otherwise the objects may undesirably flicker. In the context of computer vision, the alignment can be achieved by a correct image *registration*, which consists in localizing the camera pose in a world coordinate frame [Ans01; Kle06]. In a three-dimensional space, a camera pose has six DOF, in which three are used for the position and the remaining three for the orientation. Visual tracking attempts to find the camera pose in world space by analysing the features of sequential images.

However, augmented reality applications require real-time performance and robustness to image noise, which can be computationally challenging [Ans01; Kle06]. Consequently, prototype applications have extensively used fiducial markers (Figure 2.10), which are objects that are placed in scene and that have special geometric or color properties. Fiducial markers are easily identified and, therefore, applications use them in order to find the camera pose very quickly and solve the alignment problem [Kle06].

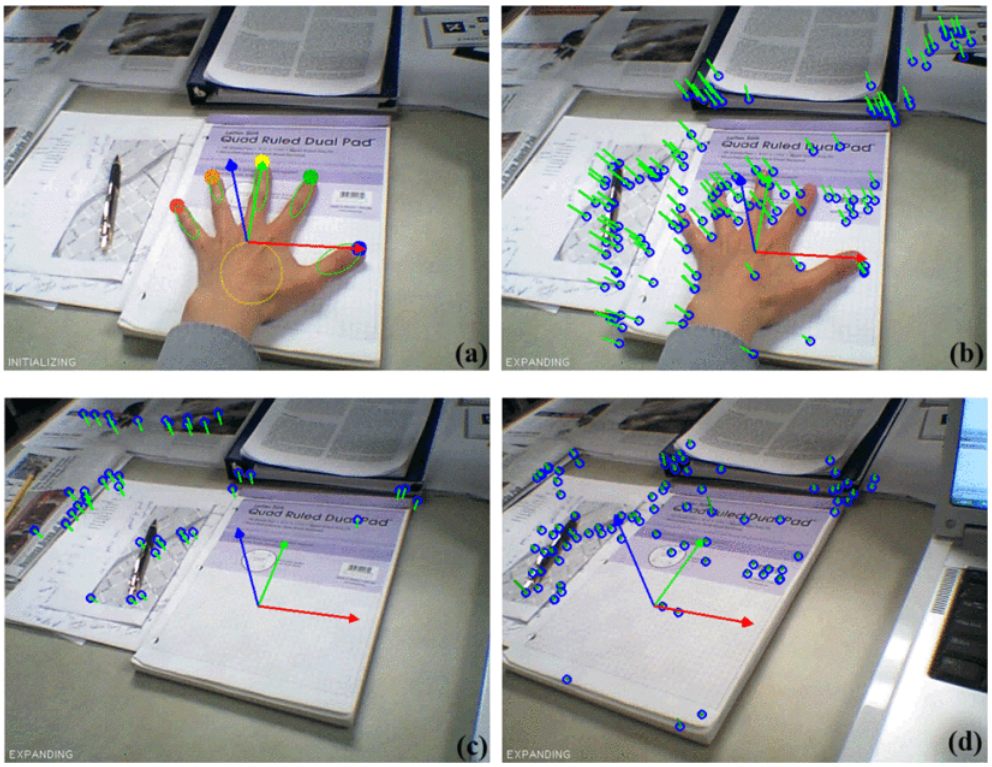


Figure 2.11: Figure (a) shows the user establishing a coordinate system with the hand, and figure (b) shows the detected features. After the hand is removed, previously occluded zones of the scene are revealed, as seen in figure (c) and new features are detected from the area that was previously occluded. Image source: [LH08].

Notwithstanding, placing fiducial objects in the scene is undesirable for non-prototype applications. Therefore, feature-based (or markerless) tracking systems, which use only features that are already present in the scene, are preferred.

The first markerless tracking systems required a priori information. That is, they required either 3D models of the scene or additional input from accelerometers [GHT11]. Another project, Handy AR, was able to estimate a camera pose relative to the user's outstretched hand [LH08]. This way, the application could establish an initial world coordinate system and use it as a reference for the origin of the scene, therefore eliminating the need to use fiducial markers, as shown in Figure 2.11.

It has also been proved that it's possible to have real-time tracking for AR without prior knowledge of the user's environment [KM07]. In the first place, a map of the environment needs to be built. This can be accomplished by estimating a dominant plane from the detected feature points of the scene, as shown in Figure 2.12. After that, the map can be used as a reference and therefore virtual objects can be inserted into the scene. In addition, the map is constantly refined and expanded if new regions are explored. However, as these are very challenging tasks, there are some limitations: the scene must be mostly static and small.

Most visual tracking techniques can be classified as either feature-based or model-based [PM06].

Feature-based tracking relies on tracking of image features such as corners or blobs. Therefore,

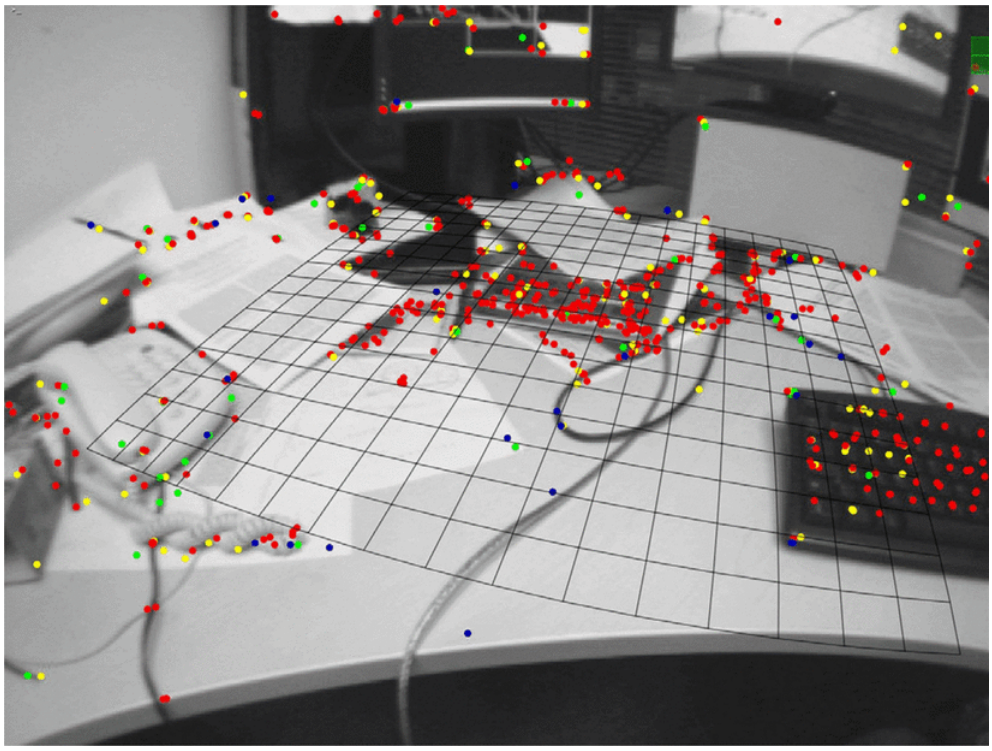


Figure 2.12: The estimated dominant plane for a desktop. Image source: [KM07].

the detection of interest points and the description of these features are the first steps of the system [GHT11]. Interest point detectors and feature description algorithms are discussed in section 2.2.1. A disadvantage of these techniques is that they are often not invariant to illumination variations [PM06].

Model-based tracking explicitly uses a model of the tracked objects, such as a 2D template of the object or a 3D model [PM06]. This class of techniques usually provide a more robust solution [PM06].

Natural feature tracking (NFT) is a feature-based technique that provides robust tracking even when the original fiducial trackers are no longer in view [ZDB08]. In the first place, the natural features present in the scene are automatically calibrated, considering the fiducial markers [PYN98]. That is, an initial camera pose is calculated from the fiducial markers. Thereafter, the system continues to acquire natural features and uses them to dynamically update the camera pose [ZDB08].

Simultaneous localisation and mapping (SLAM) refers to the problem of trying to simultaneously [Hai]:

- find the position and orientation of a sensor with respect to its surroundings;
- map the structure of the environment.

Having that in mind, a SLAM system corresponds to a set of algorithms working together in order to solve the SLAM problem [Hai]. In addition, to be considered a SLAM problem,

neither the environment nor the camera's position are known to begin with. For instance, marker-based tracking is not considered a SLAM system as the marker map is known beforehand [Hai]. SLAM systems are well suited for several purposes such as robot navigation, 3D reconstruction and markerless tracking.

2.2.3 Summary

Visual tracking is one of the most important techniques supporting augmented reality systems. The first steps of these techniques are the extraction and description of image features, which should be, along with other properties, invariant to scale and rotation and relatively immune to illumination variations. Popular robust image detectors and descriptors include SIFT, SURF and ORB. Visual tracking systems can be marker or markerless-based. In addition, they can also be classified as feature-based or model-based. NFT and SLAM systems provide feature-based visual tracking.

2.3 Technologies

Augmented reality needs a combination of several hardware and software components to work smoothly. In this section, firstly a few software technologies are analysed like game engines and AR frameworks. As one of the goals of this project is to build a prototype AR application, these technologies are compared in order to investigate which software technologies are most suited. Thereafter, hardware components such as sensors and displays are also presented.

2.3.1 Software

Nowadays, many software technologies are available to support and develop easily augmented reality applications. In this section, the main software options for developing an AR application within the context of this project are described and compared.

OpenCV

OpenCV[3] is an open source computer vision and machine learning library written natively in C++ and, in addition, it has C, Python, Java and MATLAB interfaces. It includes a set of state-of-the-art computer vision algorithms which are on the basis of many applications.

OpenCV doesn't offer a augmented reality system; although, it's possible to create one by using both OpenCV and a rendering API, such as OpenGL or Direct3D.

2.3.1.1 Unity

Unity[4] is a cross-platform 3D engine. Applications can be created by using either C#, Javascript or Boo.

[3] Opencv | opencv, opencv homepage. URL: `http://opencv.org/` (visited on 01/28/2017).

[4] Unity - game engine, unity homepage. URL: `https://unity3d.com/` (visited on 01/28/2017).

There are several plugins available that enable the creation of AR applications using Unity. In this section, some of these plugins are presented and compared.

Vuforia

Vuforia[5] is the most widely used AR platform and is available as a Unity plugin. Its major features are:

- support for a high number of platforms such as several HMD and major mobile devices;
- very robust marker-based visual tracking system;
- recognition of box- and cylinder-shaped objects;
- object recognition by extracting and tracking features of complex object geometry;
- text recognition;
- occlusion handling, which allows visual tracking of marks even when they are partially occluded by other objects;
- extended tracking which enables the application to sustain rendering, even when the marker is no longer in view;
- smart terrain technology, which provides reconstruction of a terrain in real-time, creating a 3D geometric map of the environment based on the real world objects;
- support for mono and stereo cameras.

Kudan

Kudan[6] is a SLAM AR engine. It is available as a plugin for Unity and has native support for iOS and Android. It's free to use for development; however, a license key is required if the application is submitted to a store.

Some features of Kudan that differentiate them from the other available frameworks are:

- it supports both marker and markerless visual tracking;
- robust system with low overhead and low memory footprint;
- can be used across all range of devices, such as mobile phones, HMD, robotics and embedded systems;
- can track multiple markers simultaneously, even multiple numbers of the same marker with no upper limit;
- sensor agnostic, supporting mono and stereo cameras, depth-sensors and others.

[5] Vuforia | augmented reality, vuforia homepage. URL: `https://www.vuforia.com/` (visited on 01/28/2017).
[6] Home | kudan, kudan homepage. URL: `https://www.kudan.eu/` (visited on 01/28/2017).

Figure 2.13: SLAM engine of Kudan. On the left side, the blue circles represent detected interest points, which are then mapped into a 3D environment, as seen in the right side. It's also possible to observe the world coordinate system (1), the current position (2) and path of the camera, represented in yellow. Image source: [KudanTech16].

.

ARToolKit

ARToolKit[7] is an open source AR framework that is cross-platform and is available as a plugin for Unity. The main features of this framework are:

- fiducial objects tracking using the Natural Feature Tracking algorithm;
- camera callibration support;
- support for stereo cameras;
- optimized for mobile devices.

2.3.1.2 Unreal Engine

Unreal Engine[8] is a suite of game development tools. Applications can be developed by using C++ and/or visual blueprint scripting.

However, unlike Unity, AR support is very limited. Only one plugin was found to be robust enough to be considered as a viable option for the development of this project.

Unreal4AR

Unreal4AR[9] is a plugin for Unreal Engine that enables the creation of AR applications using the Unreal Engine's blueprint visual scripting system and it's based on the ARToolKit framework. It only supports visual tracking with fiducial objects. In addition, its major drawback is that the free option for development doesn't include required engine modifications to deploy an application on

[7] Open source augmented reality sdk | artoolkit.org. URL: https://archive.artoolkit.org/ (visited on 06/26/2017).

[8] Unreal engine technology | home, unreal engine homepage. URL: https://www.unrealengine.com (visited on 01/28/2017).

[9] Unreal4ar. URL: http://www.unreal4ar.com/ (visited on 01/28/2017).

Android. A non-free personal license is required, making it impractical for use in the context of this project.

2.3.1.3 Conclusions about Software Options

One of the objectives of this dissertation is to build a prototype AR platform.

OpenCV offers computer vision algorithms that can be used to build an augmented reality system. However, the creation of an entire augmented reality system is not the focus of this project. Plus, it would probably not be robust enough when compared with already existing and available solutions.

Unreal Engine is also not suitable for this project, as there is no robust free framework available for augmented reality.

The most viable solution is to use Unity for project development as it offers countless features, including cross-platform project deployment. Three free frameworks that provide plugins for Unity are summarized in Table 2.1.

ARToolKit is a very robust framework for marker tracking; however it lacks in features when compared to the other options.

Vuforia provides a highly robust marker tracking, alongside several potentially useful features such as model recognition, partial occlusion handling and extended tracking.

Kudan uses a different technology for visual tracking, allowing both marker and markerless tracking. The SLAM system of Kudan can be helpful for the project, as the environment is mapped to a three-dimensional space, which can be particularly useful for the remote user system.

Table 2.1: Comparison between the selected frameworks which provide a Unity plugin.

Plugin Name	Marker Tracking	Markerless Tracking	Model Tracking	Occlusion Handling	Algorithm
Vuforia	Yes	No	Only some shapes	Yes	N/A
Kudan	Yes	Yes	SLAM	Yes	SLAM
ARToolKit	Yes	No	No	Yes	NFT

2.3.2 Hardware

Essential hardware components for augmented reality include sensors and displays. This section gives an overview of relevant components within the context of this dissertation.

2.3.2.1 Sensors

An AR system must be able to track the camera pose and movement. The main types of tracking techniques are sensor-based, vision-based and hybrid-based [ZDB08]. Considering that the software technologies analysed on Section 2.3.1 are all vision-based, only sensors used within visual tracking techniques are discussed.

The sensors used for vision-based tracking include mono and stereo cameras and depth sensors.

Depth cannot be directly computed from a single RGB camera. Consequently, visual tracking systems using monocular cameras have to correctly re-estimate the depth over time by analysing multiple frames.

On the other hand, stereo cameras and depth sensors provide means for a more accurate depth estimation. Therefore, these type of sensors are preferred over mono cameras. Depth sensors can use several techniques to produce depth estimates, such as *time-of-flight* imaging, *structured-light stereo*, *dense passive stereo* and *laser range scanning* [HBH+17].

A stereo camera consists of two cameras separated by a fixed distance. The depth can be calculated by triangulation, by observing the same physical point in both cameras. Alongside depth sensors, stereo cameras allow the direct estimation of the depth of a point in the scene in real-world units[10].

2.3.2.2 Displays

Visual information can be transmitted in many forms. Displays used for augmented reality can be very distinct and, consequently, they have been classified. Azuma et al. [ABB+01] divided them into three categories: head-worn, handheld and projection displays. In this section, the classification provided by Van Krevelen and Poelman [VP07] is presented, which classifies displays according to their visual display and to their position.

Visual Display

According to Van Krevelen and Poelman [VP07], displays can be classified according to the way they transmit visual information as:

- video see-through;
- optical see-through;
- projective displays.

In a video see-through display, reality is digitalized. Therefore, these displays offer much more control over the result; hence, it's possible to synchronize the virtual image with the scene before displaying, resulting in stable virtual objects [CFA+11]. Another possibility would be to remove real objects from the final image, such as fiducial markers [VP07]. In addition, the virtual objects can be adjusted in terms of contrast and brightness in order to better match real environment [VP07]. However, they also have some disadvantages. Comparing with optical see-through, the real-world is not perceived as naturally [CFA+11] and the field of view may also be limited. Also, in the case of head-worn displays, most have a fixed focus position which can result in poor eye

[10] Scale in simultaneous localisation and mapping | kudan, kudan blog article. URL: https://www.kudan.eu/kudan-news/scale-simultaneous-localisation-mapping/ (visited on 02/03/2017).

accommodation [VP07]. Systems that use video see-through displays are also more demanding than systems that use optical see-through displays, as they require two cameras [CFA+11].

In a optical see-through display, the reality is seen through the display and the virtual objects are overlaid. As the reality is not digitalized, these type of displays are safer because, in case of fail of power, the users can still see the reality, which is extremely important for military and medical purposes [VP07]. However, this displays require additional input devices such as cameras, for registration and interaction. In addition, as they use mirrors and lenses to combine the virtual objects with the real-world, the resulting images have less bright and contrast, making them less suited for outdoor use [VP07]. The field of view is also limited and the user may see the virtual objects being clipped at the edges of the display [VP07]. On one hand, systems that use optical see-through displays are less demanding than video see-through displays because they employ a mirror technology to allow the physical world light to pass through the lens and overlay virtual information to be reflected in the user's eyes [CFA+11]. On another hand, in contrast with video see-through displays, the view of the real world cannot be delayed and the time lag introduced by the graphics and image processing is perceived by the user, which may result in unstable and jittering virtual objects [CFA+11].

The projective displays technique consists in projecting the virtual objects into the real world. It has the advantage that they can cover a large area an can be seen my multiple users. However, this technique requires the use of projectors that need to be calibrated each time the distance to the surface changes [VP07].

Display Positioning

Based on their position, displays may be classified into three categories [VP07; CFA+11]:

- head-worn;
- hand-held;
- spatial.

Head-worn displays include *head-mounted displays* (HMD), *virtual retina displays* (VRD) and *head-mounted projective display* (HMPD) [VP07]. This type of displays provide a hands-free AR experience, which is of extreme importance when the user needs to perform some task in the physical world, such as performing assembly or maintenance tasks [ZBW+14].

Hand-held displays employ small computing devices with a display that the user can hold in their hands [CFA+11]. They include smartphones displays as well as hand-held projectors [VP07] and they are usually less expensive than head-worn displays. Smartphones offer a combination of high computation power and several sensors such as cameras, accelerometer, GPS and solid state compass, making them a very good platform for AR [CFA+11]. As they are extremely portable and widespread, they are suited for learning purposes. However, their small display size is not ideal for 3D user interfaces [CFA+11]. In addition, in the context of an industrial environment or

collaborative work, they may not be as suitable as head-worn displays, as they require to be held by hand.

Spatial displays separate most of the technology from the user and integrate it into the environment and are well suited for large presentations and exhibitions, with limited interactivity [VP07; CFA+11].

2.3.2.3 Summary

Monocular cameras are present everywhere; although special algorithms need to be employed in order to estimate correct depth of the scene. When available, stereo cameras or depth sensors should be used as they allow a direct computation of the depth. Displays can be classified differently relative to their visual display and positioning. As a visual display, they can be either video see-through, optical see-through or projective. Regarding positioning, they can be classified as head-worn, hand-held or spatial.

Chapter 3

AR Collaboration Framework

In this chapter, an AR framework for remote collaboration is described and discussed. Subsequently, an implementation of the framework, as well as its architecture are specified.

3.1 Collaboration Framework Proposal

To enhance the remote collaboration process between users, an AR-based framework is proposed. Specifically, this framework allows the creation of complex instructions by the remote user. It also allows remote collaboration without sending too much information from the local user's environment. This may be a concern in cases where increased network latency is a problem or when privacy is a must. The system used by the remote user also allows the saving of previous collaborations sessions, in order to allow quicker sessions in the future.

3.1.1 Requirements

To elaborate the collaboration framework, some requirements were defined, considering the goals established for this dissertation. These include requirements of: the collaboration process, the system, the user and augmented reality.

The framework must provide a client-server architecture, with two different clients: one to explain the tasks (mentor's client) and the other to present the instructions using augmented reality (apprentice's client). The server may be located on the mentor's system. The two clients must be running simultaneously and have access to a network connection in order to initiate a collaboration session and communicate. The mentor's client may be used independently for preparing tasks, as long as it has access to the server.

Ideally, the apprentice's client should be deployed on a system that provides an AR hands-free experience. In the context of remote collaboration, a major number of tasks may require the apprentice to use their hands. Therefore, head-worn displays should be preferred over hand-held

displays, as they provide a hands-free experience, which can help the apprentice to perform the given tasks.

The apprentice's client should provide a very robust visual tracking in order to accurately display the augmented reality instructions in their correct places. Ideally, visual tracking could be done by using a marker to identify the scale and origin of the world and use markerless-based tracking from that moment on. This way, users could use the system and move the camera freely without worrying about the marker being occluded or going off-screen.

Regarding the users, no special requirements are needed to use either application. However, users that possess a good perception of the 3D space will find it easier to use the mentor's system in order to prepare the tasks and organize the instructions in a 3D environment. The design of the platform's interface should have into account users with certain disabilities that still may be able to use the system. For instance, although the system may provide communication by voice, it should allow the completion of collaboration tasks without requiring the use of any sound, considering users with hearing loss.

3.1.2 Framework Description

The framework encompasses two subsystems that communicate with each other. As seen in Figure 3.1, the mentor prepares instructions that are sent to the apprentice. These instructions are presented using augmented reality that guide the apprentice performing the task.

One of these subsystems is used by an apprentice that needs to perform a complex task over a certain object, within the physical world. The other subsystem is used by a mentor that aims to help the apprentice performing the task. The apprentice's subsystem uses augmented reality to aim the user performing the required operations, whereas the mentor interacts with a virtual environment in order to freely explore the objects in focus and instruct the apprentice as required.

The virtual environment which the mentor interacts with has key similarities with the apprentice's real environment. Specifically, the objects that are being used to perform the task by the apprentice are in some way represented in the virtual scene of the mentor. These objects can be created by the mentor itself using tools provided by the application.

This framework proposes to structure information hierarchically in three levels:

1. task: a piece of work that can be divided into several steps, e.g., *how to change a bicycle tire*;

2. step: a simple action, e.g., *reattach break*, and is composed by instructions, animations and state of the scene objects;

3. instruction or hint: something that can provide useful information for the task at hand; they may be text, 2D images, 3D objects, or even the action of highlighting a certain real-world object.

Animations can be used to emphasize certain instructions and aid the mentor to transmit an idea, e.g., *to convey motion*.

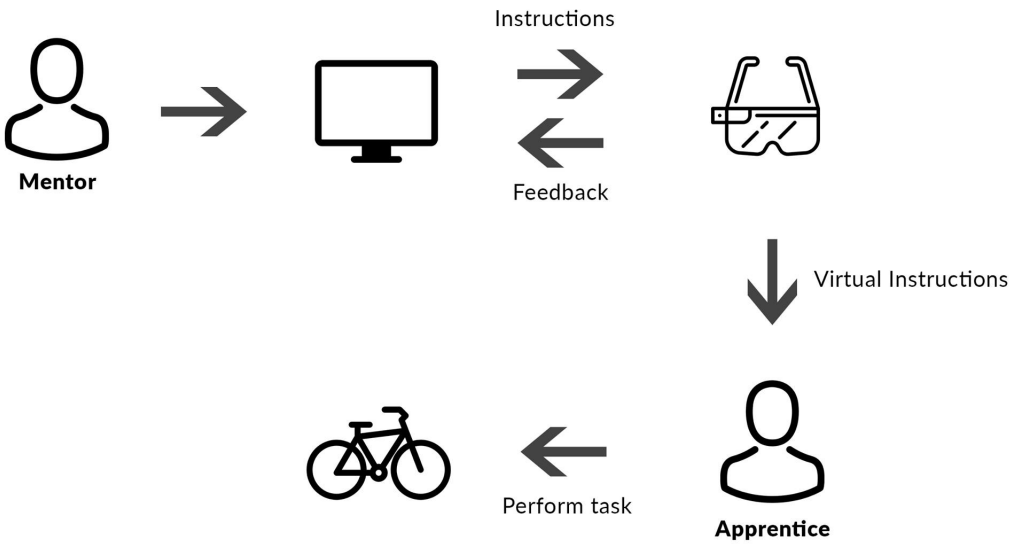


Figure 3.1: Interactions in the AR collaboration framework.

The mentor's subsystem offers several sample instructions, such as arrows and geometric primitives. The creation of custom instructions should also be supported. New instructions may be created by the mentor, by importing custom images and meshes.

Figure 3.2 displays an UML use case diagram showing the possible interactions with the system. The mentor is able to create, edit and delete tasks, steps and instructions. The mentor is also able to navigate freely through the virtual environment, in order to better study the objects in focus and find the best position to give appropriate feedback. Objects in the mentor's environment may also be modified or hidden in a particular step. This is useful for two reasons: to accompany changes in the apprentice's environment (e.g., *the apprentice removes a wheel from a bicycle*); and to give a better view for the mentor, as some objects may occlude others and difficult the mentoring task.

The apprentice is able to send feedback back to the mentor, either by interacting with the interface or by sending a video stream, depending on the implementation. The interface provides visual cues to guide the user if there are instructions that are out of the field of view.

Before beginning a collaboration session with the apprentice, the mentor needs to prepare the task, define its steps and, for each step, create the corresponding instructions. Then, when the task is set and selected, the apprentice may connect with the system and initiate a collaboration session.

The general workflow of a collaboration session is shown in Figure 3.3. The instructions and animations belonging to the first step of the selected task are presented to the apprentice using augmented reality. Then, the apprentice performs a certain action according to the instructions.

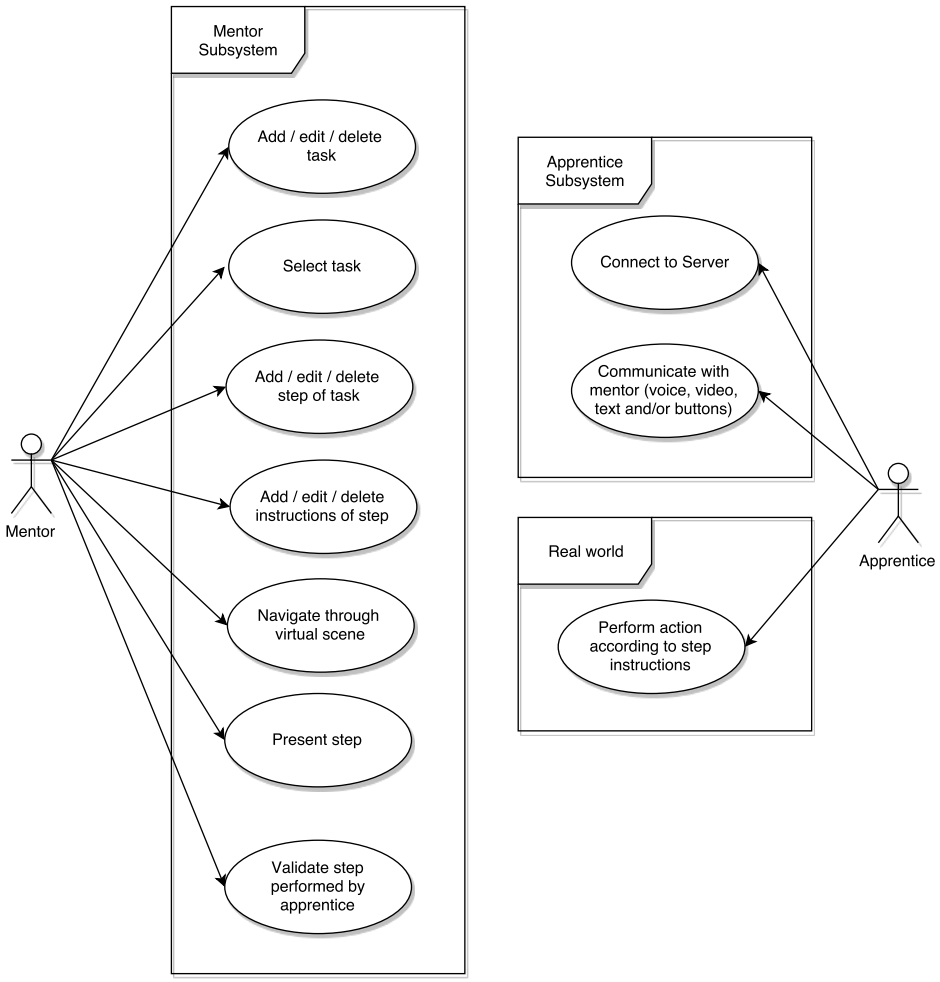


Figure 3.2: UML use case diagram.

In some way, feedback from the apprentice needs to be sent to the mentor. This can be done in several ways and depends on the implementation. One possibility is to use a video stream displaying the environment of the apprentice. Another possibility is to send a photo of the task environment from time to time (e.g. at the end of each step). Another way would be to use text messages or buttons (e.g. a button for informing that the step was completed). This approach would greatly reduce network bandwidth; however, it would considerably limit the capacity that the mentor has in validating the actions performed by the apprentice.

If the apprentice had not been able to accomplish the step and needs help to proceed, the mentor would interact with apprentice until the action is successfully completed, by, for instance, creating new instructions or making them more evident. As before, the implementation may provide multiple ways of interaction, such as voice calls or text messages. When the apprentice is able to complete the step correctly, the mentor presents the next step and so on, until the all steps are concluded.

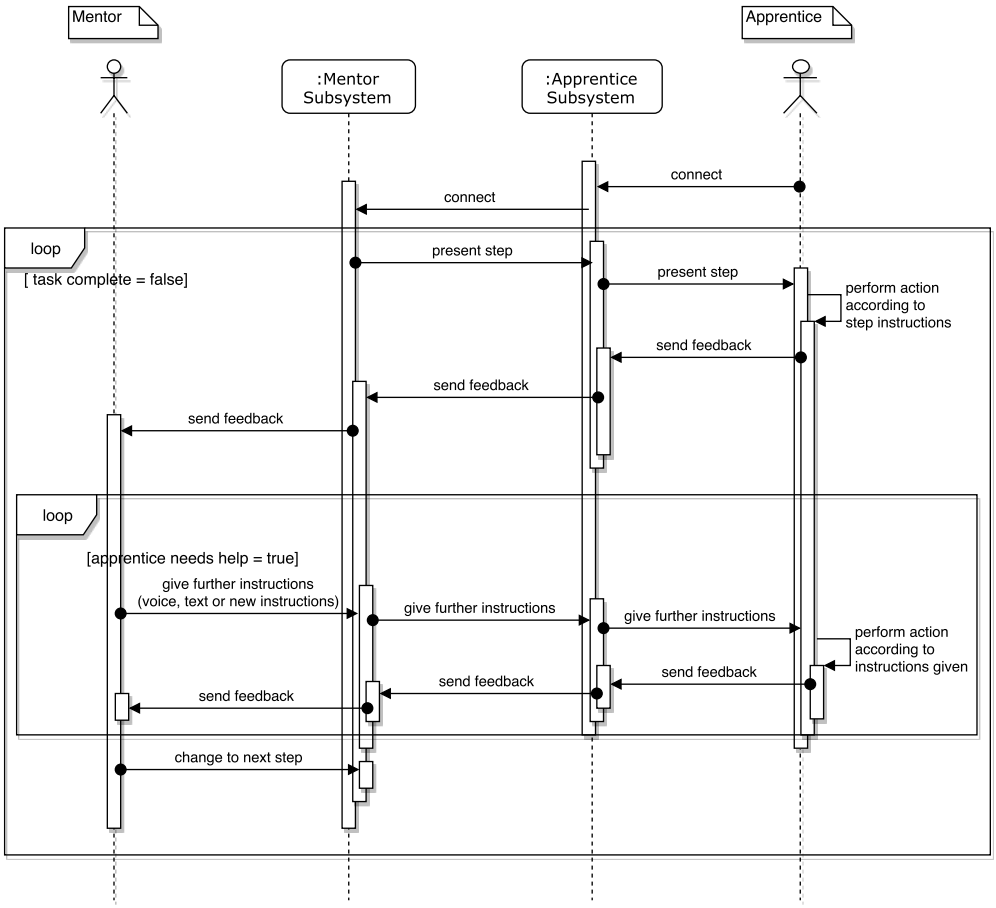


Figure 3.3: UML sequence diagram.

3.1.3 Discussion

On most remote collaboration systems, the mentor has a limited view of the apprentice's environment. An advantage of this framework, is that it allows the mentor to freely explore the apprentice's environment through a virtual scene, easing the task of giving instructions by providing free movement of the virtual camera. Another advantage is that tasks created from previous collaboration sessions can be saved for future use.

A great development effort should be put on the feedback module, as it should allow the mentor to validate if the step was successfully completed. A video stream captured by the apprentice and sent to the mentor would enable the proper validation of steps. In addition, it should allow communication between apprentice and mentor, so that question may be asked and answered back, by, for instance, allowing communication by voice.

One of the major limitations of this framework is that the mentor's scene has to be created before hand. Still, it can be useful to create collaboration systems in which the same environment is repeatedly used, such as in machine maintenance. In the future, this framework may be extended in order to support the creation of the mentor's scene in real-time using a 3D reconstruction technique on the apprentice's side and sending the collected data to the mentor's subsystem.

One good example that enables real-time 3D reconstruction and interaction is the KinectFusion system [IKH+11].

3.2 CooP AR

Based on the framework described, a platform named CooP AR (Collaboration Platform in Augmented Reality) was developed. This platform consists in two applications that implement the subsystems of the mentor and the apprentice. The mentor's and the apprentice's applications communicate with each other, the server being the mentor's application itself. The mentor's application was developed targeting Windows, while the apprentice's was deployed on Android.

3.2.1 Requirements

CooP AR, as an implementation of the collaboration framework, shares many requirements of the platform. However, it also has different requirements, some due to the current technology and others related with the available equipment.

Regarding the requirements of the system, the apprentice's application needed to be deployed on a device running Android with a camera and access to a network connection.

With respect to AR, it was required that the apprentice's application provided a robust visual tracking system, so that virtual instructions would not flicker or disappear during the execution of the tasks, compromising the efficacy and efficiency of the collaboration sessions. In addition, the computing processing power of the AR module could not be be very high, as the target system to deploy was a smartphone. It was allowed the use of AR markers in order to help the stabilization of the visual tracking; however, the tracking needed to be robust enough to allow partial occlusion of the markers once detected. The markers themselves needed to be small enough in order to be fit onto the smartphone's camera frame and, at the same time, allow the user to perform the given tasks.

3.2.2 Mentor Application

In this application it is possible to:

- create, edit, delete and select tasks;
- manage steps of a particular task;
- manage instructions of a step, which can represent text, images or geometry;
- present a different step;
- change between several cameras;
- control each camera individually;
- receive notifications regarding the apprentice's activity.

3.2.2.1 Initial Menu

In the first place, the mentor is presented with an initial menu (Figure 3.4) in which the collaboration tasks can be managed. It shows a list of previously created tasks.

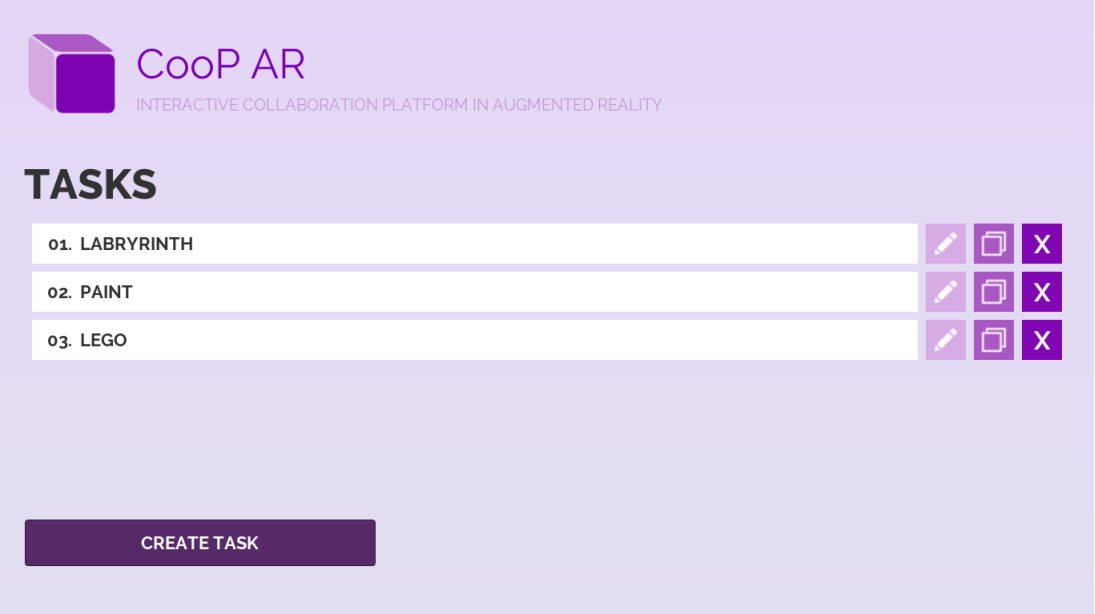


Figure 3.4: Initial screen of the mentor's application.

To create a new task, the user clicks on the respective button and a new item appears. The user needs to give a name immediately and only then the task is successfully created. To select a task, one simply clicks on the the respective name.

By clicking on the edit button, the user may change the name of the task. The duplicate button, as the name suggests, duplicates a task completely (not only the name, but also the steps and instructions of the respective task). It is also possible to delete a task.

3.2.2.2 Editor

The editor screen (Figure 3.5) displays a 3D viewport and several tools for navigating, managing steps and instructions and to communicate with the apprentice. By clicking on the top left icon labeled *CooP AR*, the application goes back to the initial menu. This editor has three fundamental parts: the cameras' views panel, the instructions panel and the apprentice box.

A common problem when positioning objects on 3D software is the optical illusion that an object is located just on the right position, when, in fact, it is far from the intended position. To effectively transform an object in 3D space, it is useful to switch rapidly between several points of view. To help with this issue, the camera's panel, located on the left side of the screen, can be used.

The camera views available are aligned with the front, back, top, bottom, right and left sides of a virtual cube at the origin. By default, these cameras use an orthographic projection, and,

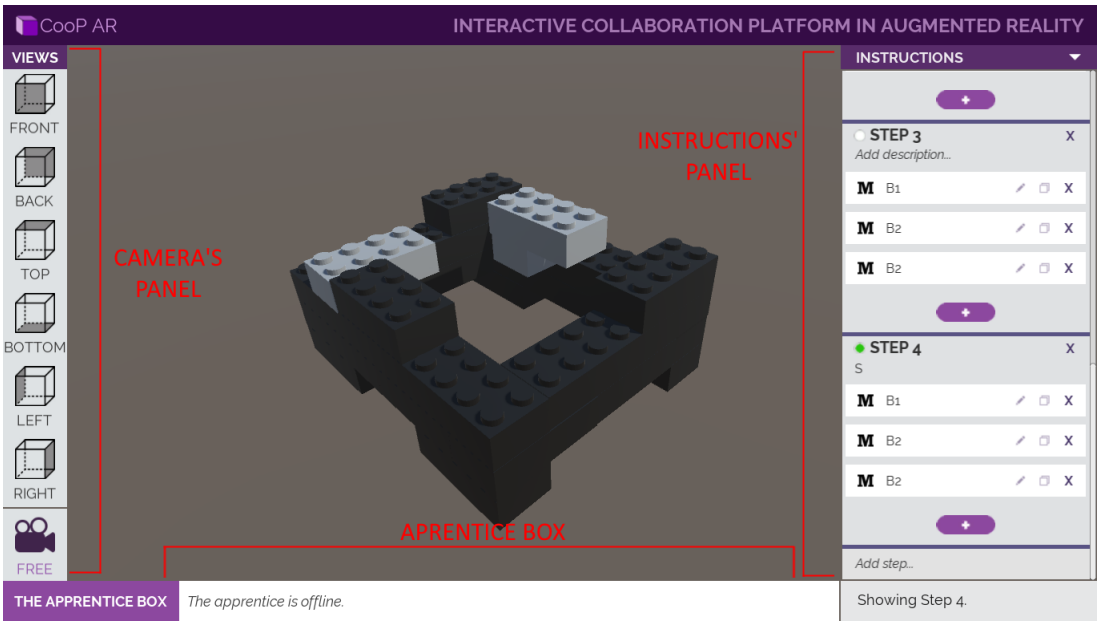


Figure 3.5: Editor screen.

consequently, the projection lines remain parallel to each other (i.e. parallel lines are not distorted by perspective). This effect is particularly useful for doing accurate transformations and rotations.

In addition, a free camera which can be moved and rotate as needed is available. By default, this camera uses a perspective projection and, therefore, distant objects seem smaller than nearby objects.

To select a camera, the user simply clicks on the respective icon, which becomes purple to flag it as the currently selected camera. In the case of a side camera, the user may use the *W*, *A*, *S*, *D* keys to move the camera up, left, down or right, respectively. A side camera will only move within its respective plane. If a camera uses an orthographic projection, it is possible to zoom in and zoom out using the mouse wheel. The zoom in/out effect is achieved by decreasing/increasing the vertical size of the viewing volume, since the horizontal viewing size varies depending on viewport's aspect ratio. The free camera moves forward, left, backwards and right by pressing *W*, *A*, *S*, *D*, respectively. By pressing the right button of the mouse, it is possible to rotate the camera freely. To further control the camera rotation, it is also possible to use *Q* and *E* to roll the camera counterclockwise and clockwise, respectively.

To control the sensibility of the movement, rotation and zoom of the cameras some parameters were defined. These parameters can be tweaked for each task, as different tasks may be processed at different scales.

These parameters as well as other camera properties such as position, rotation and projection type are unique to each task and they are persistent. That is, the properties are saved after a session and, for instance, a camera will have the same position and orientation as in the session before. This allows the mentor to prepare a task for future use, by tweaking the properties of each camera only once, easing future use.

The apprentice box displays useful information relatively to the apprentice's activity. On its default state, the box displays whether the apprentice is online or offline. The message is changed when the apprentice transmits that the step was completed or that more instructions are needed. In the case of the completion of the step, the box returns to display its default message after the mentor changes to another step. If the message was that the apprentice needs more instructions, the box changes its message to the default after the mentor makes any change to the instructions. Whenever the message is changed, a notification sound is played to make the note explicit.

The instructions panel (Figure 3.6), located at the right side of the screen, is used to manage the steps of the given task.

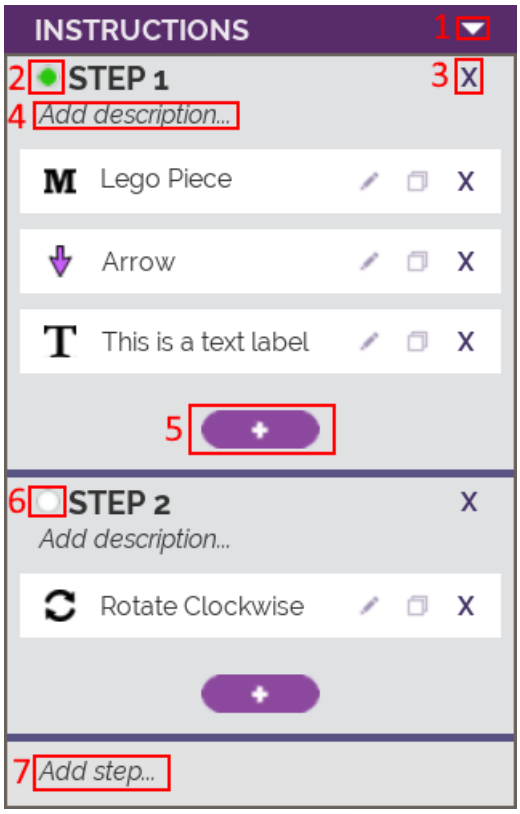


Figure 3.6: Instructions' panel. (1) Hide toggle. (2) Show step 1 toggle. (3) Delete step. (4) Description of the step. (5) New hint button for step 1. (6) Show step 2 toggle. (7) Add new step.

The first time an empty task is selected, a new step is automatically added on the instructions panel. The user may fill a description if desired. The description of a step is used merely for organizational purposes from the point of view of the mentor, as this information is never transmitted to the apprentice. As the instructions panel can take a lot of space on the screen, it may be desired to temporarily hide it by clicking on the respective toggle button.

To add a new step, the user would click on the *Add step...* message and fill in a description of the step. After clicking on *Enter*, a new step with the given description is created. For easing certain tasks, the hints from the last step are cloned. The user may then decide to edit, clone or

delete these hints. Deleting a step can be achieved by simply clicking on the *X* at the top right of the corresponding step panel.

A green circle beside the step name indicates that step is currently selected. The message on the right bottom of the screen also points out which step is being shown. This message provides greater clarity, as the instructions panel may be collapsed or the step may be concealed on the scroll view. When the apprentice is connected, the selected step is presented to the apprentice through augmented reality. To select another step, the mentor must click on the toggle button of the respective step.

Each step can have several hints. An hint is represented by both an item on the instructions panel and by a 3D object. The item (Figure 3.7) on the instructions panel provides buttons to edit, duplicate and delete an hint.

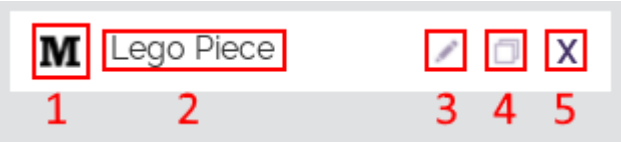


Figure 3.7: Hint item. (1) Hint type icon. (2) Hint name. (3) Edit button. (4) Duplicate button. (5) Delete hint button.

In this application, there are three types of hints: text, image and geometry. The text hints are useful for transmitting certain messages through 3D text. The 3D message is equal to the name given to the hint. An image hint spawns a 2D square with a given texture. Geometry hints are used to spawn 3D models, such as a cube, a sphere or a lego piece.

By clicking on the + button associated with a step, a new panel pops up (Figure 3.8) in order to add a new hint to the corresponding step. The name field should be filled with a description of the hint. Then, the user needs to select the hint's type. In the case of selecting the image or geometry type, the panel expands to show new options. For instance, as can be seen on Figure 3.8, the window expand so that it is possible to select the geometry. After clicking on *OK*, the window closes and it switches to edit mode.

While on edit mode (Figure 3.9), it is possible to move, rotate and scale the 3D representation of the hint by using a transformation gizmo. In addition, it is also possible to change the name of the hint. This is particularly useful for changing the value of text hints. While on this mode, the user cannot create, duplicate or delete new hints and steps. To exit the edit mode, the mentor needs to click on the edit button, which is colored in purple.

During edit mode, the user may transform the 3D hint by using the transform gizmos (Figure 3.10): translate, rotate and scale. The top white panel can be used to switch between gizmos, as can be seen on Figure 3.9.

Each gizmo is composed by three components colored differently: red, blue and green. In the case of the translate and scale gizmos, the lines red, blue and green correspond to the *X*, *Y* and *Z* axis, respectively. In the case of the rotate gizmo, the red, blue and green circles are used to rotate the object around the *X*, *Y* and *Z* world axis, subsequently.

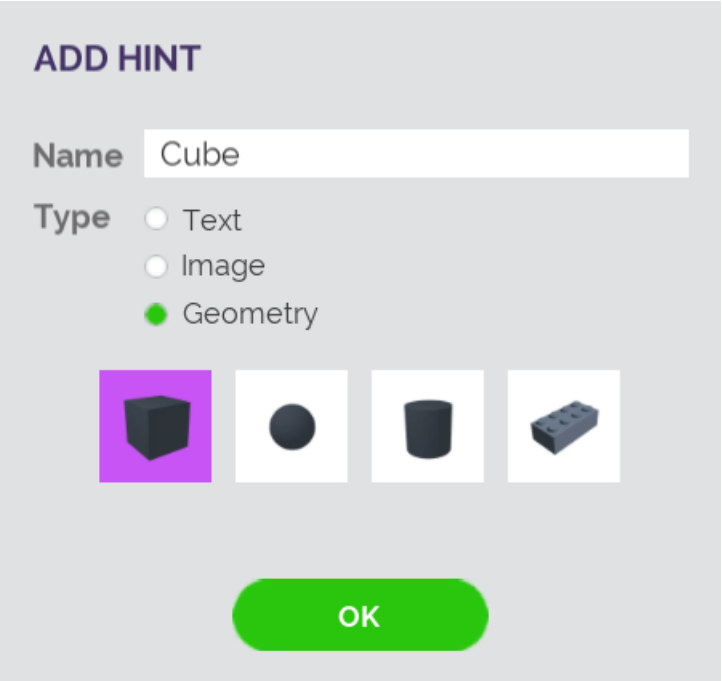


Figure 3.8: Add hint panel.

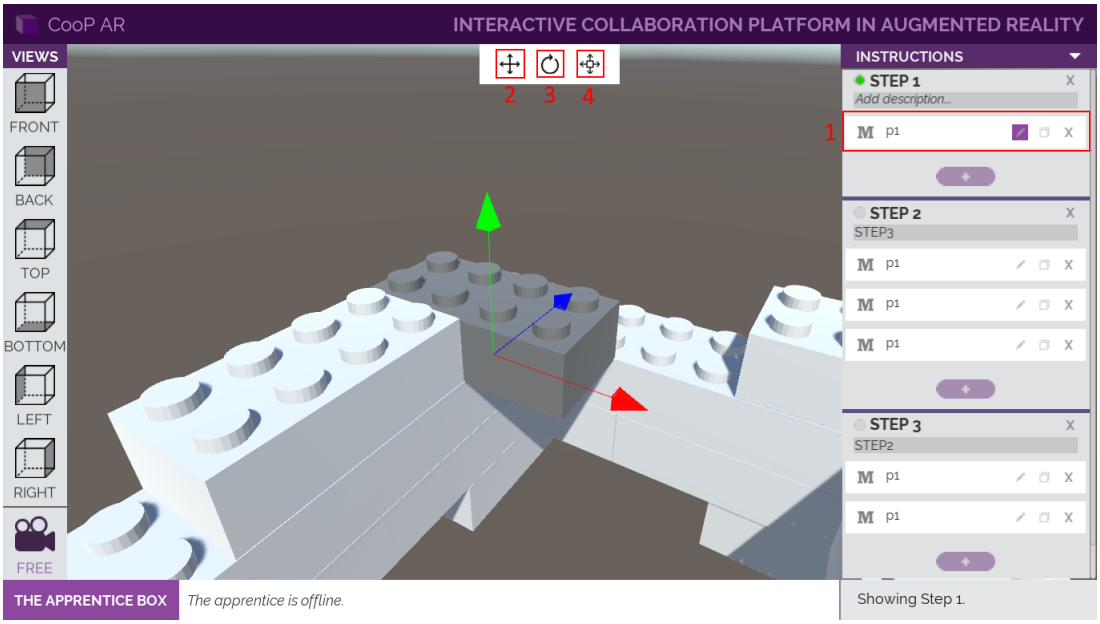
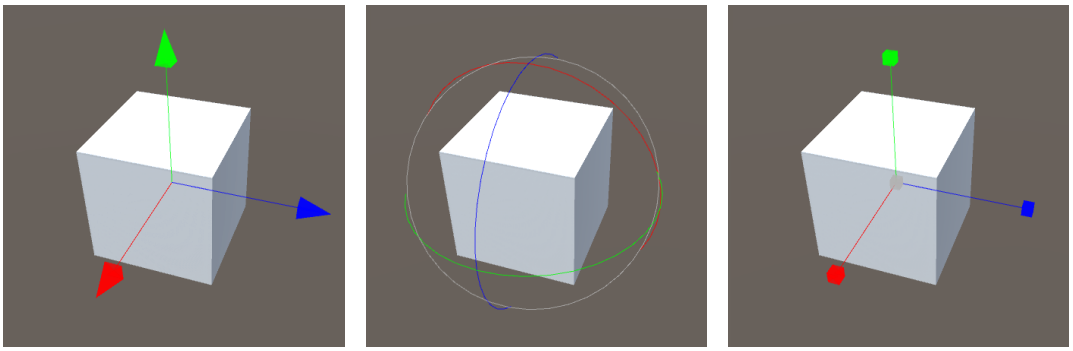


Figure 3.9: Edit mode. (1) Hint being edited. (2) Translation gizmo toggle. (3) Rotation gizmo toggle. (4) Scale gizmo toggle.



(a) Translate. (b) Rotate. (c) Scale

Figure 3.10: Transform gizmos.

In order to use the gizmos, the user needs to click and drag a component, which will change its color to yellow to convey that it is selected. The scale gizmo contains a central square which can be use to scale the object uniformly. The transform gizmo used is an adapted version of the Unity3DRuntimeTransformGizmo [RGizmo].

3.2.3 Apprentice Application

The apprentice application targets the Android operating system and requires a camera, in order to use the augmented reality features. The apprentice application allows to:

- enter the IP address of the server and initiate a connection;
- focus a certain marker using the camera and visualize the AR instructions;
- inform the mentor that a step was completed;
- inform the mentor that the instructions are not clear and need to be further explained.

The apprentice application begins by displaying an initial screen, in which the user needs to enter the IP address of the mentor device. As the mentor application acts as a server, it must be already running before trying to establish a connection.

After establishing a connection with the server, the apprentice application switches to an augmented reality collaboration mode (Figure 3.11). On this screen, the hints of a certain step created using the mentor application are displayed using augmented reality. When the mentor switches for another step, the AR instructions change accordingly.

The top label is present to help the apprentice keep track of the progress and to make it more explicit that the mentor changed a step. During the process, if the apprentice feels that the instructions for the given step are not clear enough, clicking on the *?* button will send a notification to the mentor application, so that the instructions are changed. When the apprentice completes the step, clicking on the button at the bottom right of the screen will inform the mentor that the step was completed and that the next step, if it exists, should be selected.

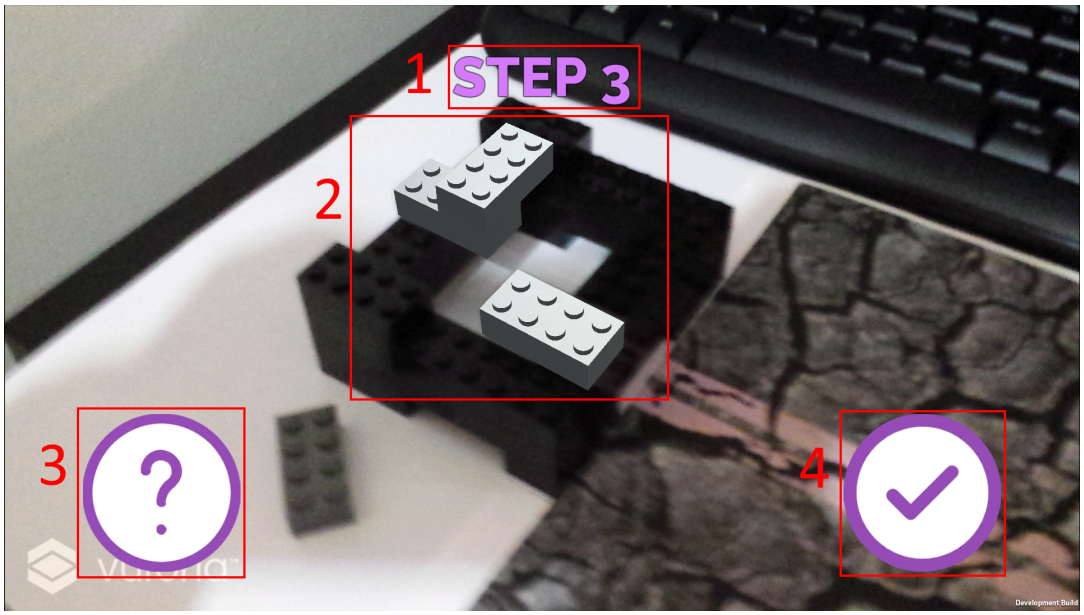


Figure 3.11: AR collaboration screen. (1) Current step label. (2) Augmented reality instructions. (3) Need more instructions button. (4) Step completed button.

This application uses fiducial markers to display the AR instructions, and they may be different for different tasks. For a particular task, the apprentice needs to first let the application recognize the respective marker. During the process, the user needs to hold the camera in a way so that the marker is always partially visible. If the visual tracking is lost, the application displays a message informing that the target was lost and it is trying to reacquire it.

3.3 System Architecture

The architecture of the application developed was based on the model-view-presenter (MVP) architectural design. This pattern is typically used to separate the internal logic (model) from the user interaction (view) through the use of an intermediary (presenter or supervising controller) [mvp]. The model is completely oblivious of the other two components, and raises events when the state of data changes. The view represents information and raises user events that are processed by the controller. The controller subscribes to the user events of the view and state-change events of the model, and modifies the view and the model as appropriate.

Figure 3.12 displays an high-level representation of the general architecture of the platform and the interactions between modules. Following the MVP pattern, the logic is divided into three different modules. Both mentor and apprentice interact with their application's respective view. User events are redirected to the controller, that updates the model accordingly.

Figure 3.13 gives an overview of the mentor's application architecture in more detail. All the information about the hints, which are annotations that correspond to the third level of information described in the Section 3.1.2, are represented by the hint model class, which is described in more detail in Section 3.3.1. Having in consideration the *Hint Model*, the *Hint Controller* is able to

create a 3D view of the annotation, which can be rendered as a 3D geometry, an image or text. To transform the annotations in the 3D space, the user interacts with the *Transform Gizmo*, which raises *transform events* that are redirected to the *Hint Controller*. The *Hint Controller* updates the model and sends the updated data to the apprentice's application, through the *Network* module. The actions performed by the apprentice (e.g. when it informs that a step was completed) are received by the *Network* module and redirected to the *Apprentice Box* module that updates the notification message. In addition, the mentor is able to see a 3D static world that is specific for each task, and control cameras to navigate through the scene. The *Cameras* module is described in Section 3.3.4.1.

Figure 3.14 displays the apprentice's application architecture in more detail. The *AR Controller* module receives data from the mentor's application through the network. This data can correspond, for instance, to all information that is needed to render all hints of the presented step by the mentor. The received data is used by the *AR Controller* module to update the view, by creating 3D representations of the annotations (hints), that are displayed using augmented reality. All the interactions of the user are captured by the *AR Controller* and sent, through the network, to the mentor's application.

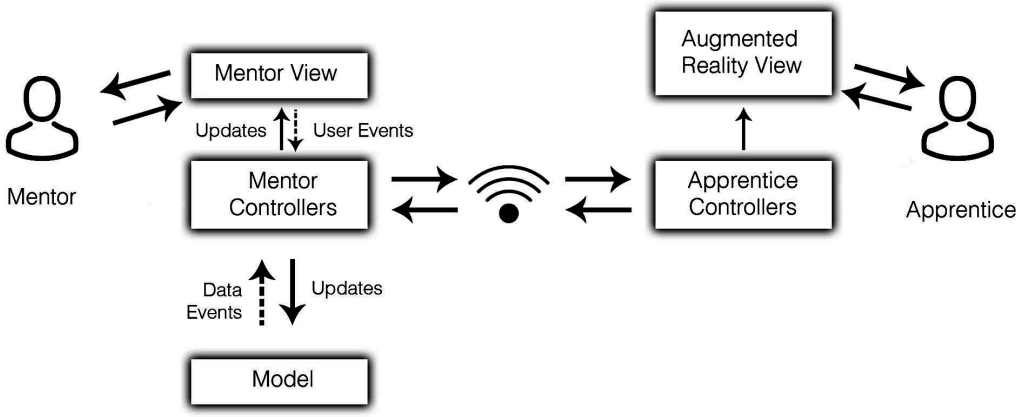


Figure 3.12: High-level overview of the platform and interaction between the different modules.

Unity, the platform used to develop this project's applications, uses an entity-component system [gdec13]. In the case of Unity, every entity is derived from the *GameObject* class. To add behaviour to each entity, different components are attached. Every game object is composed by a *Transform* component, which in conjunction with other *Transform* components allow the formation of a scene graph, named Hierarchy in Unity. Unity also allows the creation of *prefabs*, which are predefined game objects previously created using the editor, that can be spawned at any time in run-time.

In practice, there are many variants of how the MVC pattern is implemented. To apply the *MVC* pattern to the Unity environment, some guidelines provided by Costa [Cos15] were followed. Figure 3.15 shows the setup of the hierarchy of the game objects in Unity. An *Application* game object was created that has the *Model*, *View* and *Controller* as children. Each child has different

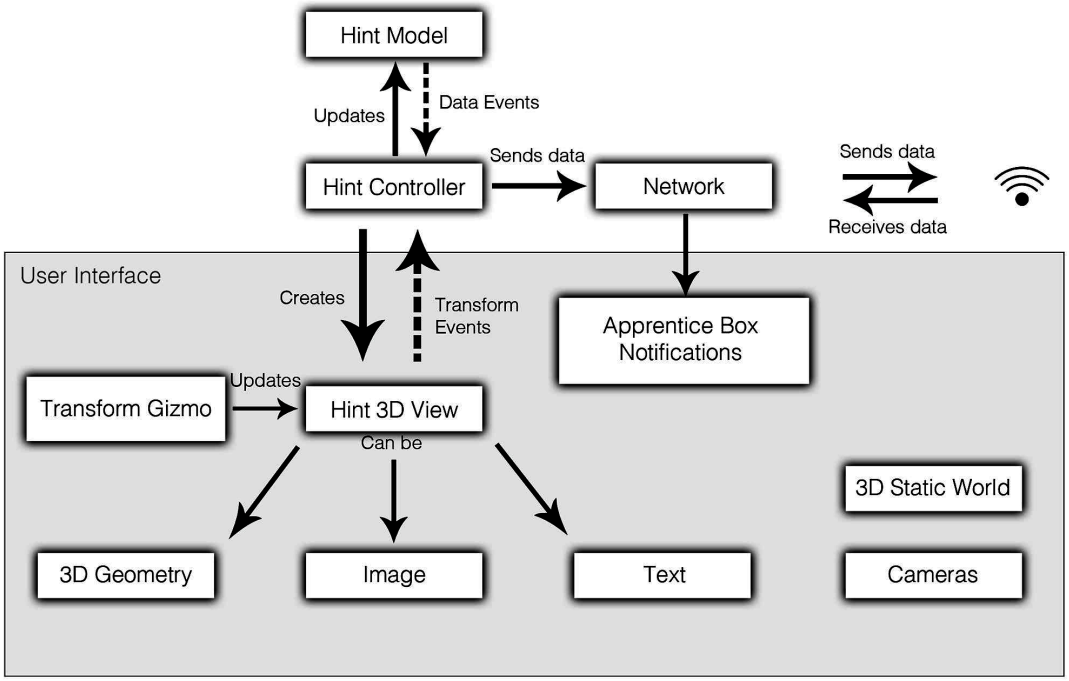


Figure 3.13: Interaction between the several modules that compose the mentor's application.

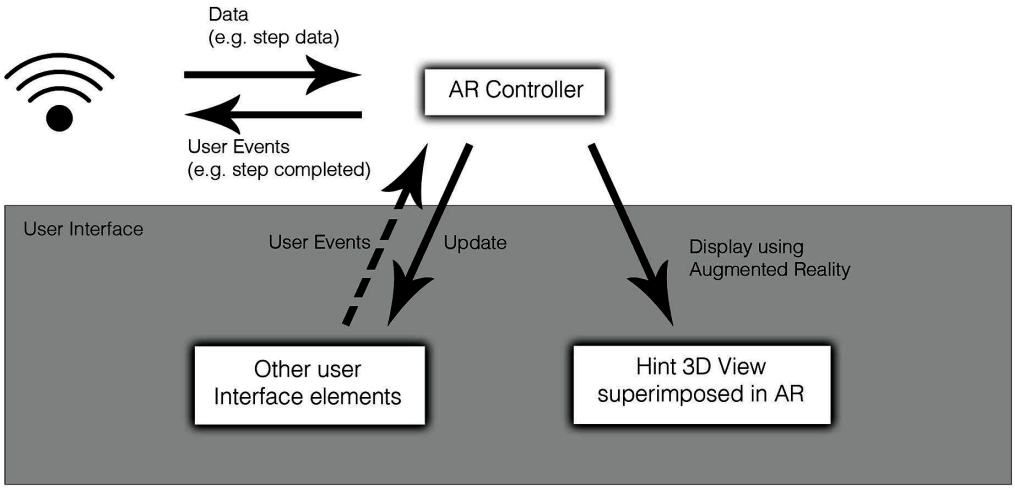


Figure 3.14: Interaction between the several modules that compose the apprentice's application.

children associated with the corresponding component.

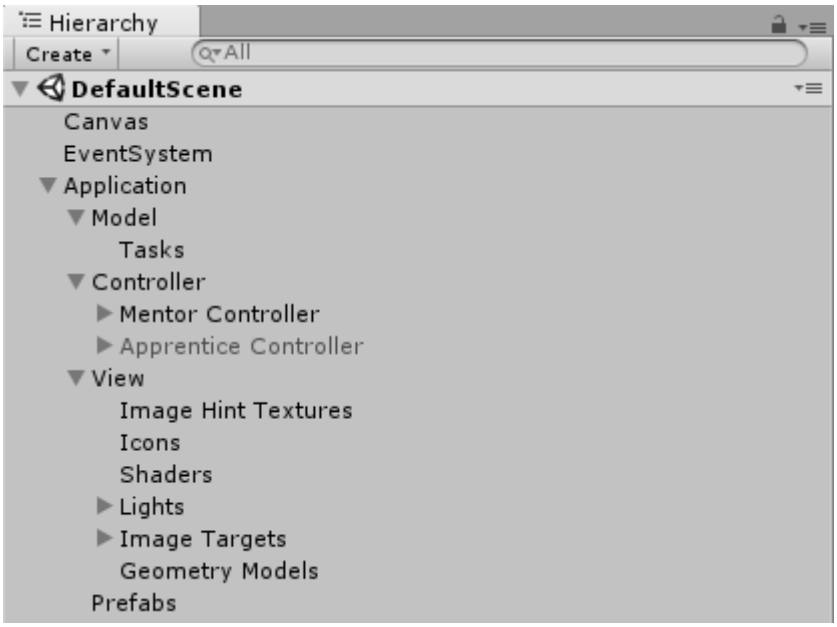


Figure 3.15: Hierarchy of the game objects in Unity.

To add custom behaviours to the game objects, Unity requires the creation of classes that derive from *MonoBehaviour*.

Figure 3.16 displays how the various components of the MVC were implemented. An *Application* class owns references to the *Model*, *View* and *Controller*, and *Prefabs* objects. In addition, as the same code is used by both applications to maximize code reuse, it also provides properties that can be used in run-time to test whether the application targets the mentor or the apprentice.

An *Entity* class derived from *MonoBehaviour* was created, that provides a property to retrieve the *Application* instance, allowing every class that derived from *Entity* to access the *Application* instance without invoking the *Singleton* design pattern. This allows, for instance, a controller to modify the tasks owned by the *Model*, modify the elements of the *View* or to spawn a certain *prefab*, through the *Application* property.

3.3.1 Model

To represent the information defined by the collaboration framework, three different classes were created: *TaskModel*, *StepModel* and *HintModel*. A class named *TasksModel* was created in order o manage all the tasks as collection. Figure 3.17 shows the relationships between these.

TasksModel contains a set of *TaskModel* (property Tasks) and provides methods to create, delete, duplicate and access tasks. In order to instantiate *TaskModel* objects, Unity requires a reference to a prefab. This field is shown on the Unity's object inspector and allows the programmer to define which prefab should be used. In the case of a *TaskModelPrefab*, the prefab is composed by a game object which contains a *TaskModel* component.

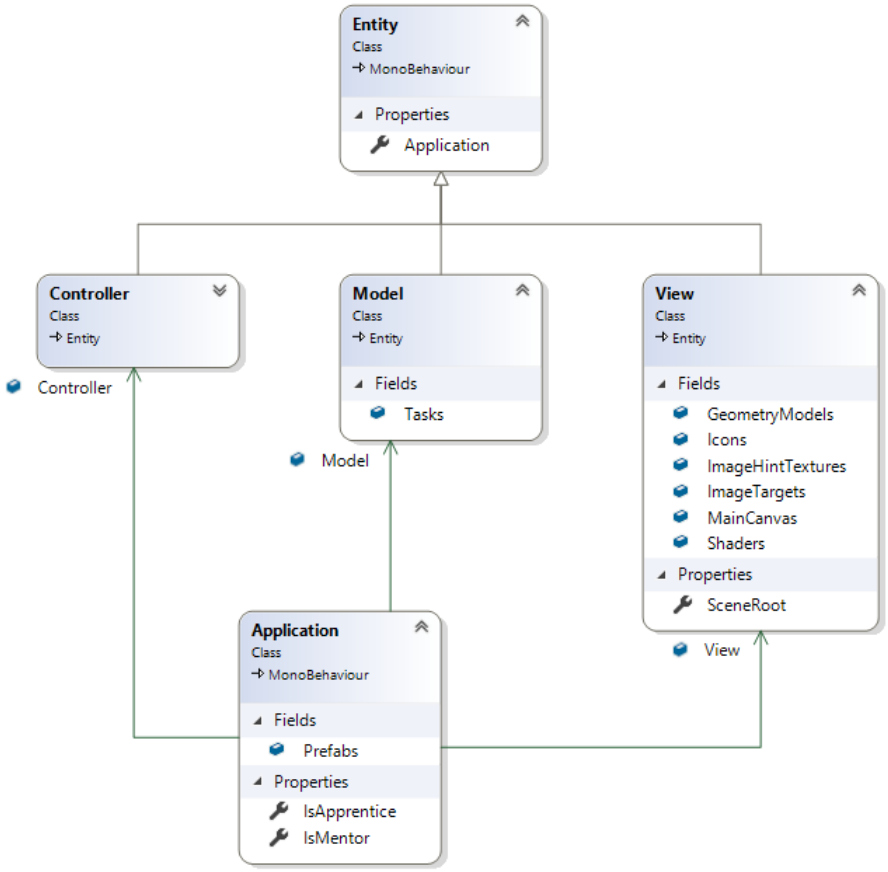


Figure 3.16: Class diagram of the classes that serve as the basis of the MVC implementation.

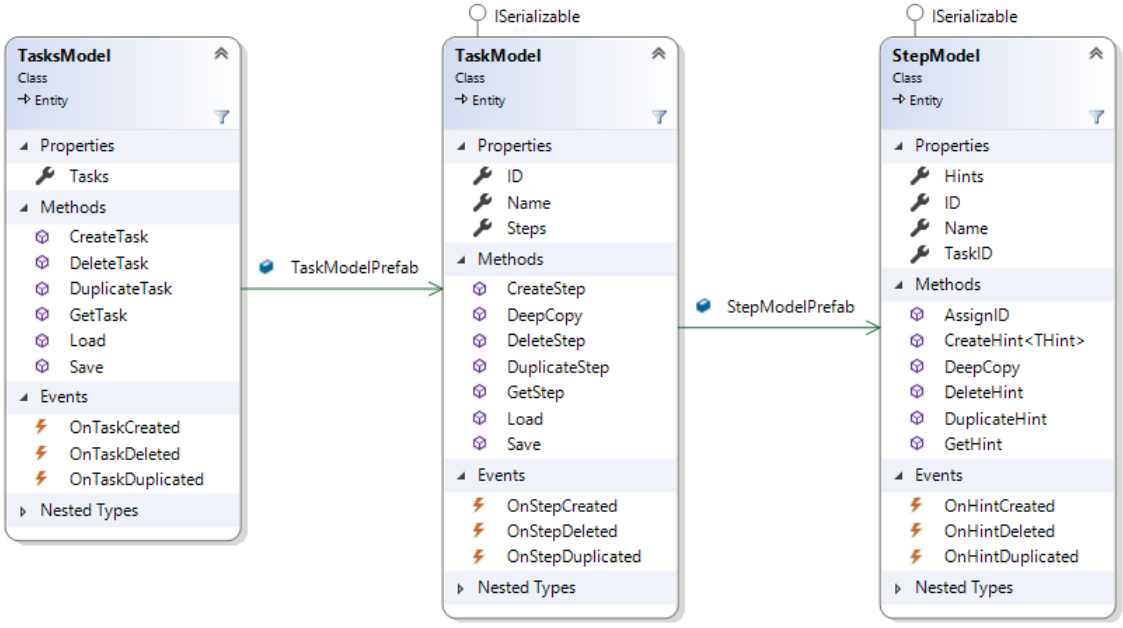


Figure 3.17: Class diagram that displays the relationship and content of the *TasksModel*, *TaskModel* and *StepModel* classes.

Each task, step and hint contain an unique identifier (ID) which is used by several functions to reference a certain item. For instance, the method *TasksModel.GetTask* retrieves a *TaskModel* object with the same ID passed as argument.

Each *TaskModel*, *StepModel* or *HintModel* have a name. In addition, *StepModel* and *HintModel* objects have IDs to their parent objects, in order to allow navigation from child to parent through the information hierarchy.

Both *TaskModel* and *StepModel* classes implement the *ISerializable* interface, which is a custom defined interface that provides the methods *Serialize* and *Deserialize*. These methods require *System.IO.BinaryWriter* and *System.IO.BinaryReader* objects as arguments, respectively. These methods provide a convenient way to write/read objects to/from binary streams. For instance, they are used to pack objects and send them through the network, when the applications need to communicate.

The methods *Load*/*Save* are used to load/store the information of the model when the application opens/closes and also use the *Serialize* and *Deserialize* methods. The data is stored in files according to the path *Saved/Tasks/{task_id}/Task.data*, in which *{task_id}* is the ID of the task. This ensures that all changes to the model in run-time are saved and restore in future runs. It also allows backups of data, by simply copying the folder *Saved*.

The methods *DeepCopy* provide a way to duplicate an item and all is children. For instance, a call to *DeepCopy* of a *TaskModel* object returns an instance of the same type, with the same fields of the original object, as well as all its *StepModel* objects, and the corresponding *HintModel* objects of each step. However, to maintain consistency, new IDs are assigned to each duplicated item.

Each class also provides a set of events which can be subscribed from any part of the code. An event is a feature of the *C#* programming language that implements the *Observer* design pattern, and allows classes or objects to notify others when something interesting happened. For more information about events please refer to *C# Reference (event)* [csevents17]; regarding the *Observer* design pattern, please refer to Nystrom [Nys17].

One particular useful case is for a view to subscribe to events of the model. For instance, a view representing a list of tasks can subscribe to the *TasksModel* events and be notified when a task is created, deleted or duplicated in order to update the view's list.

The third level of information defined by the collaboration framework is the instruction or also named hint. As the instructions could be represented in several ways, several classes derived from the base class *HintModel* were defined: *ImageHintModel*, *TextHintModel* and *GeometryHintModel*. Figure 3.18 presents this idea in a clearer way.

Each *HintModel* type has in common the *LocalPosition*, *LocalRotation* and *LocalScale* properties. These properties, as the names suggest, are used to store the values of the local transform of each instruction. Local values are used to allow different parent transforms in the apprentice and mentor applications.

Hint models are created through the generic method *CreateHint* of *StepModel*. Then, the specific properties of the returned instance can be set, such as name or the *ModelID* in the case of

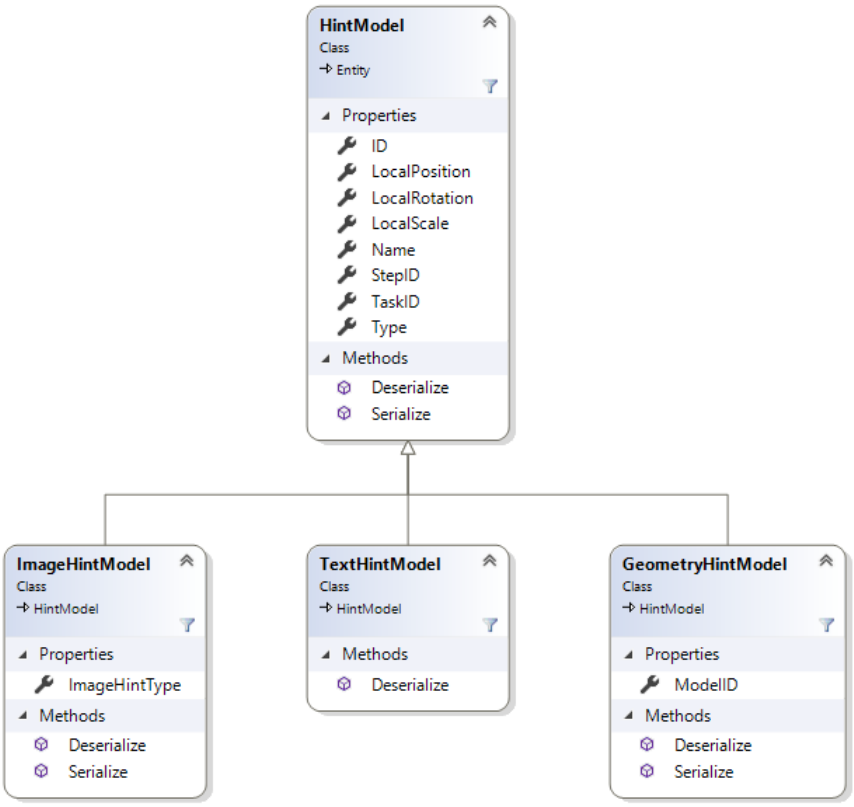


Figure 3.18: Class diagram that shows the *HintModel* classes.

a *GeometryHintModel*.

ImageHintModel objects contain an enum value (*ImageHintType*) that identifies the image that it refers to. This value can be used to fetch the image texture from *View.ImageHintTextures*. *TextHintModels* do not contain more properties, as the text value used corresponds to the *HintModel*'s *Name* property. The *ModelID* value of *GeometryHintModel* objects is used to fetch the corresponding 3D models from *View.GeometryModels*.

As *HintModel* as several derived classes, the serialization and deserialization is more complicated. In the first place, before calling the *HintModel.Serialization* method, the property *Type*, which is an *enum* value that depends on the type of hint, is written to the binary stream. Then, the method is called; however, as the method is virtual, the derived classes extend the base method and write more values (e.g. *ImageHintType* in the case on the *ImageHintModel)*. To deserialize, the *Type* value is read from the binary stream. Then, an object is instanced according to the Type value. Finally, the *Deserialize* method is called and, as before, the common and specific properties of each class are read in the overridden *Deserialize* methods.

3.3.2 Network Communication

The mentor's application provides a server that enables the communication between the mentor's and the apprentice's applications, using the Unity's UNET features. As no dedicated server is used, the mentor's application is named the *host*, as it provides a local client as well as a server. The apprentice's application acts as a remote client. This scheme can be seen on Figure 3.19.

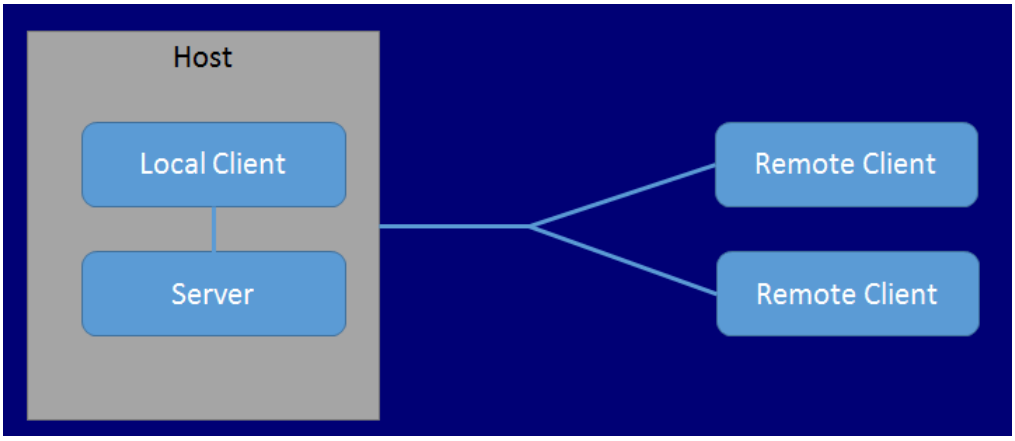


Figure 3.19: Network communication between a host and remote clients in Unity's network model. Image source: [unetcon17].

Figure 3.20 shows the main classes developed that are involved in the network communication process. The *ServerController* class is a script which is attached to a game object object that is child of the *MentorController* object, which is seen on the Figure 3.15. As such, this script is only activated on the mentor's application. Figure 3.21 displays the inspector's properties on the Unity editor. As the *Server Only* button is checked, the game object only exists on the server side. All the rest of the mentor's application game objects are part of the local client.

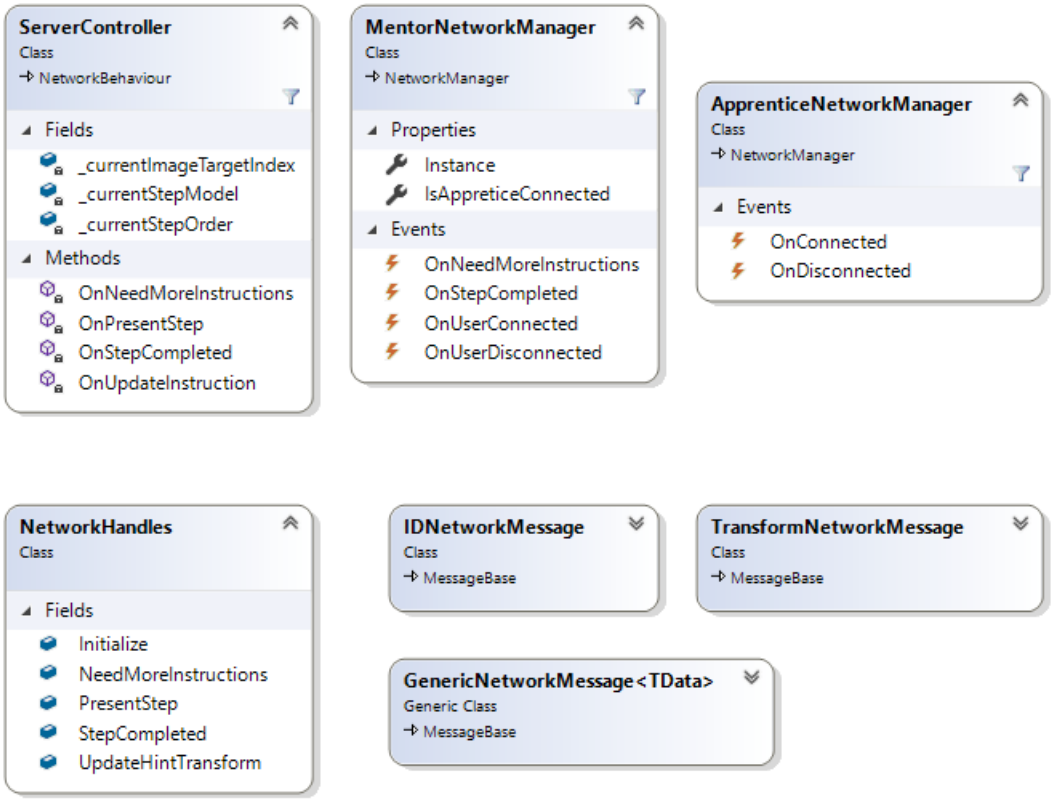


Figure 3.20: Class diagram depicting the network related classes.

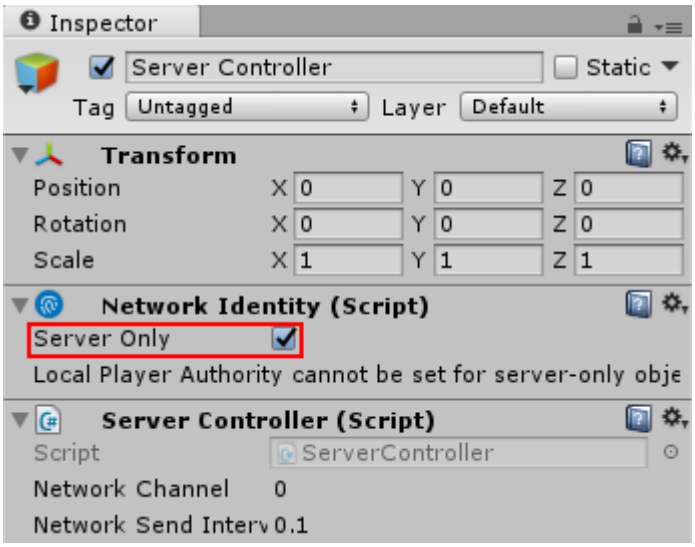


Figure 3.21: Unity's inspector view of the *ServerController* game object.

The *MentorNetworkManager* script, as the name suggests, is responsible for managing the mentor's network communication and only exists on the mentor's local client. On the other end, the *ApprenticeNetworkManager* only exists on the apprentice's remote client. Both managers can detect certain network events, such as when a connection is initiated. The view responsible for the apprentice's box, for instance, subscribes to the *MentorNetworkManager*'s events in order to display a message when the apprentice connects or disconnects.

The messages sent through the network vary on the content. *IDNetworkMessages* carry only one ID, whereas the *TransformNetworkMessages* store information about the local position, rotation and scale of an hint identified by an ID. The *GenericNetworkMessages* are used to store any kind of object, as long as it implements the *ISerializable* interface. For instance, *GenericNetworkMessages* are used to transport *StepModel* objects.

In order to allow communication between the clients and the server, the scripts need to subscribe to certain network events specified by the values of the *NetworkHandles* class. For instance, in order to send a message associated with a *PresentStep* handle:

1. the *ServerController* and the apprentice's application need to subscribe to network events associated with the *PresentStep* handle;
2. a mentor's script sends a network message associated with the *PresentStep* handle;
3. the message is received by the *ServerController* on a method bound to that purpose, that broadcasts the message to all clients which subscribed to *PresentStep* events;
4. the apprentice's application receives the message, as the apprentice's application subscribed to the network events associated with the *PresentStep* handle.

The *Initialize* handle is used to show the first instructions in the apprentice's application, is sent by the mentor's application when it detects that a connection was established and the message carries the initial *StepModel* object and its corresponding *HintModels*. The *PresentStep* message carries a *StepModel* object as well and is used, as the name suggests, to show different instructions associated with a step to the apprentice. The *UpdateHintTransform* is sent every time the mentor transforms an hint, and the data is stored in a *TransformNetworkMessage*. The *StepCompleted* and *NeedMoreInstructions* messages are sent by the apprentice's application when the user taps on the corresponding buttons. The messages carry an ID that identify which step the message refers to.

3.3.3 Augmented Reality Setup

The apprentice's application uses the *Vuforia* framework [Vuforia] to provide augmented reality features. The *Vuforia* framework was introduced on Section 2.3.1.1.

The augmented reality technology used by *Vuforia* requires the detection of targets that identify the transformation of game objects in the real world. In order to specify which targets need to be detected, *Vuforia* requires the creation of a target database using a web application named *Target Manager*. The database accepts several types of targets: image targets, cuboids, cylinders

or even 3D objects. In the case of this project, the database is composed by three different images, that were specifically added to be used for the experiment. Future work could aim to overcome this limitation and allow the mentors to add targets to the database dynamically.

The database was downloaded from the *Target Manager* web application and added to the Unity project. For each image target, a new object on the hierarchy is created, containing a script that associates the database and target to the object. Figure 3.22 displays the location of the image target game objects on the hierarchy. Each image target object contains a *Scene Root* child intermediary. When an image target is detected by the apprentice's application, the corresponding child objects are shown through augmented reality, except if they contain the *HideInApprentice* custom script. The *HideInApprentice* is attached to objects so that they appear on the mentor's viewport, while not being rendered by the apprentice's application, as the objects should already be represented in the real world.

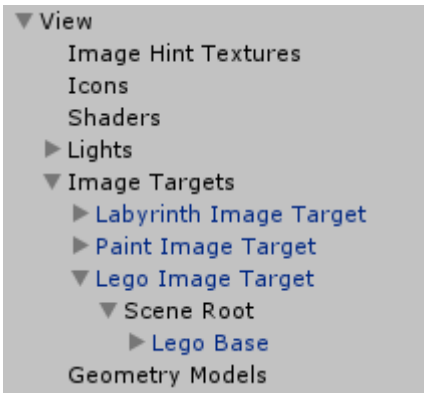


Figure 3.22: Hierarchy of the View game object displaying its children, specifically the image targets.

Each *TaskModel* object stores a value that identifies which target is used. When instructions views are created, these are added as children of the corresponding image target object. The creation of the image targets, as well as the association between tasks and image targets is only possible using the Unity editor. Future work could also focus on implementing an user interface to allow this setup.

3.3.4 Mentor System Architecture

Figure 3.23 describes relevant controllers essential to the application. The *MentorController* script is the starting point of the mentor's application. As the name implies, it acts as a controller in the *MVC* architecture. It starts by calling the method *PresentStartScreen* which instantiates the *StartMentorController*. As it owns a reference to the view's prefab, it spawns the view and the start screen is displayed. The *OnTaskSelected* event is subscribed by the *MentorController*, so that when the user selects a task, it is notified. When the event is raised, the *MentorController* destroys the *StartMentorController* and invokes the *PresentEditorScreen* that instantiates the *EditorController* and displays the view in the same manner. The *OnGoBack* event is subscribed in order to

switch back to the start screen when required. It can be noted that the *EditorController* owns a reference to a *InstructionsController* object and to the selected *TaskModel*.

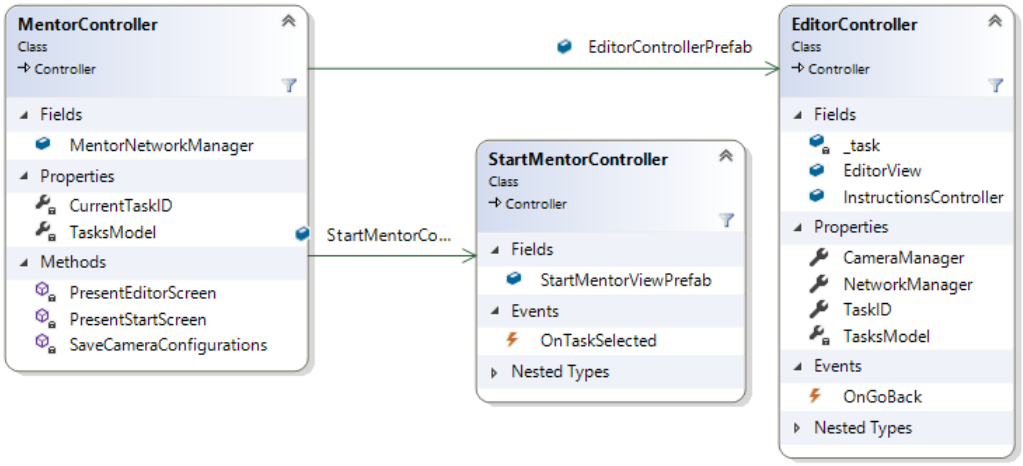


Figure 3.23: Class diagram displaying the main controllers of the mentor's application.

Figure 3.24 describes the *InstructionsController* and the *TransformPanelController* which have key responsibilities on the editor's screen logic.

The *InstrucionsController* manages the the instructions panel (Figure 3.6). It starts by creating the *StepControllers* for each *StepModel* owned by the selected *TaskModel*. It also subscribes to the selected *TaskModel*'s events (e.g. *OnStepCreated*, *OnStepDeleted*) in order to keep the panel updated. In addition, it is also notified by the *NetworkManager* that a connection with the apprentice's application was established, to send an *Initialize* message containing the currently selected step to the apprentice's application. The events of every *StepController* that is created are subscribed, in order to take appropriate actions. For instance, when the show button of a step is clicked, the instructions of the previously selected step are hidden, the instructions of the current step are shown, the message on the bottom left bar is updated to show the correct step number and a *PresentStep* message is sent to the apprentice's application.

When the edit button of an hint item is clicked, the *TransformPanelController* is spawned. This controller instantiates a panel which is used by the mentor to switch between the translate, rotation and scale transform gizmos. When the user clicks on a button, this controller simply changes the type of the transform gizmo to the corresponding value.

Figure 3.25 depicts the controllers of steps and hints. Similarly to the *InstructionsController*, the *StepController* starts by creating a *HintController* for each existing *HintModel* child of the corresponding *StepModel*, and to subscribe to the *StepModel* events. When the ad new hint button is clicked, it is responsible to instantiate a *NewHintWindowController* which handles the window that is to used to specify the properties of the hint. When the window is closed, the *StepController* is notified and it creates a new *HintModel* object based on the information filled by the mentor. As the same class subscribes to the *StepModel* events, a *HintController* is created on the fly. It also

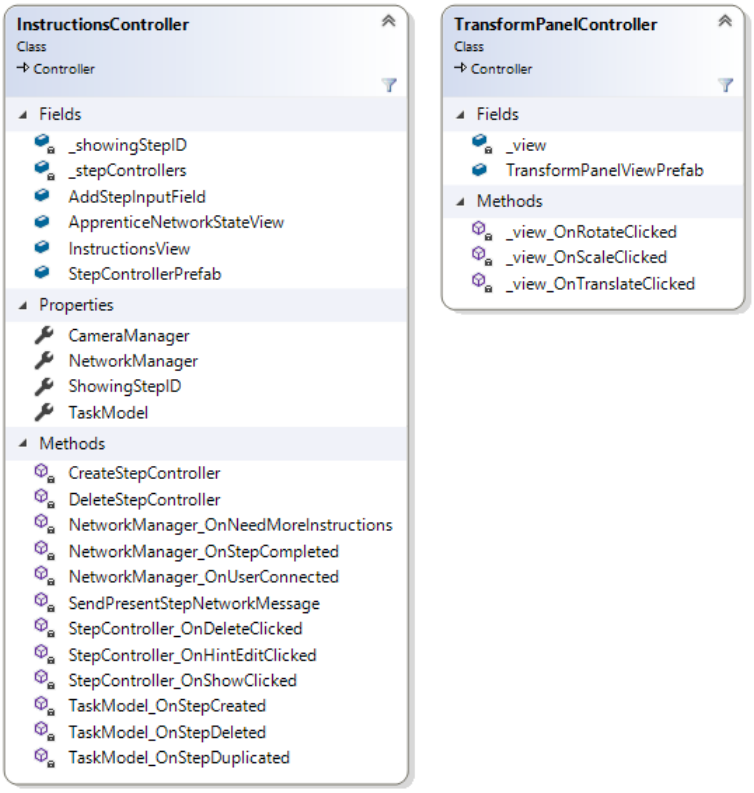


Figure 3.24: Class diagram showing the main controllers of the editor's screen in the mentor's application.

orients the hint in order that it is aligned according to the camera view. It also triggers a function so that the application enters in edit mode automatically, as soon as the new hint window is closed.

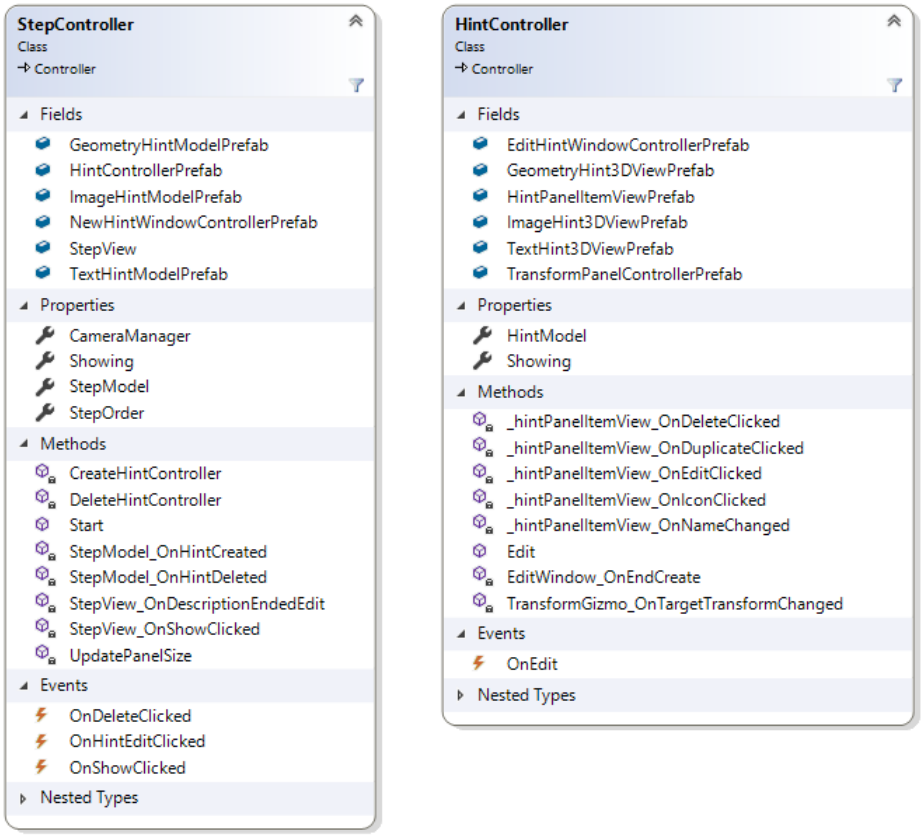


Figure 3.25: Class diagram that describes the *StepController* and *HintController* classes.

A *HintController* is responsible to create two different views for the same *HintModel*. One of the views is the hint item on the instructions panel that allows the user to edit, duplicate or delete it. The other view is the 3D representation of the instruction on the viewport. The *HintController* updates the *HintModel* when required (e.g. when the name of the hint is changed or when the user transforms the 3D view using the transform gizmo). In addition, when the hint is transformed, it sends a *UpdateHintTransform* message to the server that redirects it to the apprentice's application.

Figure 3.26 describes how the views of the 3D instructions are implemented. The derived classes share the properties that describe the transform of an object and the *Showing* property changes whether the hint is rendered or hidden. Each derived class possess a specific property that allows them to represent the instructions in its own way.

3.3.4.1 Cameras

The mentor's application allows the user to switch between seven different cameras. Figure 3.27 shows how the camera game objects are organized on the hierarchy. It can be noted that the cameras are children of a *Camera Manager* object, which is subsequently a child of the *Mentor Controller* game object.

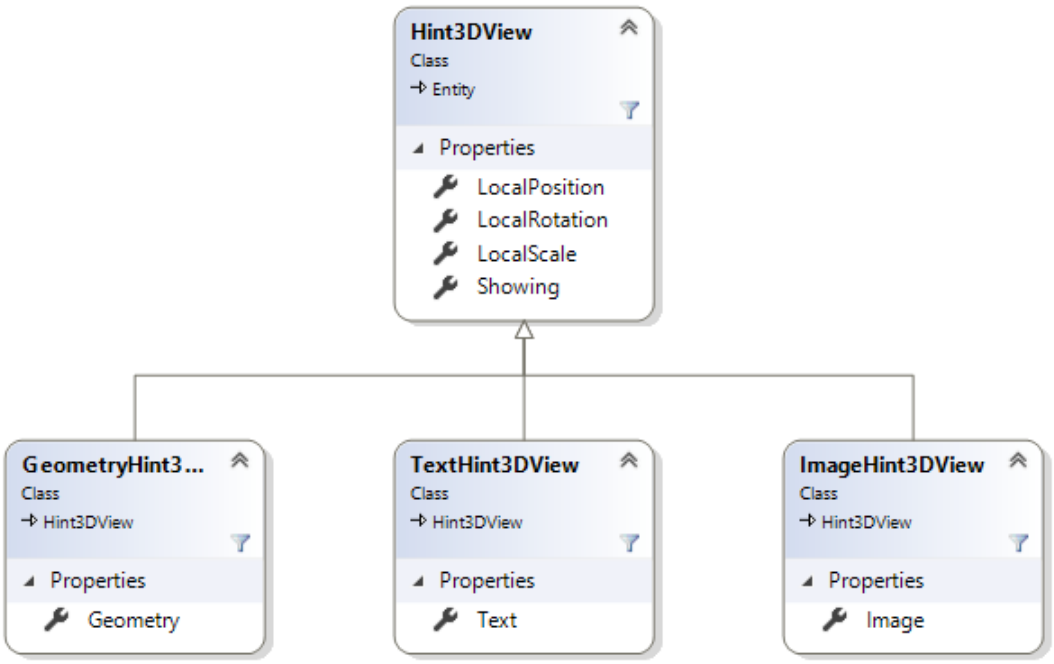


Figure 3.26: Class diagram showing the classes that represent the 3D view representations of the instructions.

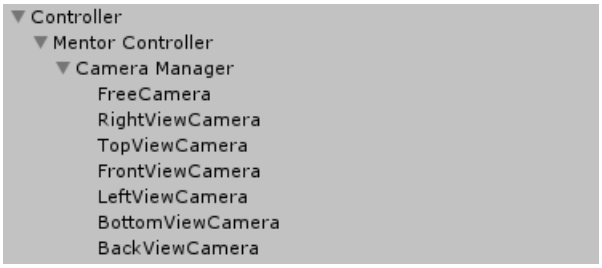


Figure 3.27: Hierarchy organization of the cameras' related game objects.

Figure 3.28 displays the classes related with the control of the cameras. There are two type of cameras: *FreeCameras* and *SideCameras*.

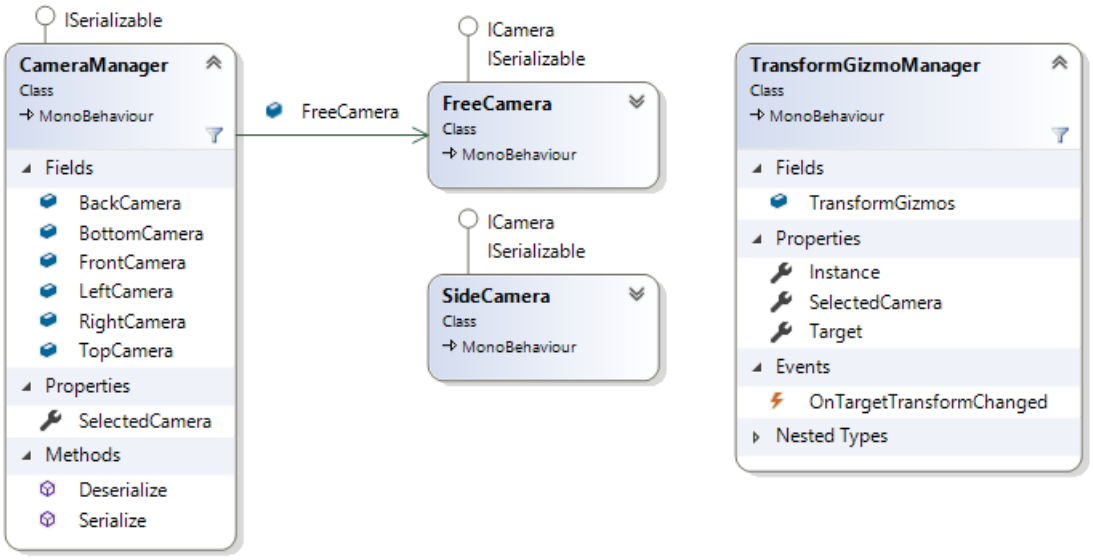


Figure 3.28: Class diagram of the camera related classes.

The *FreeCamera* was designed to move around freely, without any restriction allowing movement and rotation in every direction. The *FreeCamera* rotation is handled by quaternions, which has the advantage of not suffering of the gimbal lock problem present on Euler angle based cameras. The gimbal lock is a problem that occurs when two rotation axis become parallel, leading to the loss of one degree of freedom.

A *SideCamera* possesses a view direction and allows movement only on the plane perpendicular to that view direction. It allows zooming, but it does not allow any kind of rotation.

Each camera type is composed by sensibility parameters that can be adjusted for each task separately. These parameters include scalar values for movement, zoom, tilting and mouse sensibility. These parameters can be adjusted in run-time using the Unity editor and are loaded/saved when a task is opened/closed. Future work could also be focused on implement a user interface to adjust the cameras' parameters values without requiring the Unity editor.

The *CameraManager* class owns a reference for one *FreeCamera* and for six *SideCameras* with different viewing directions. The implementation of the *TransformGizmo* is dependent on the camera's view, thus each camera game object also contains a *TransformGizmo* script. When the mentor selects another camera, the *CameraManager* ensures that the *TransformGizmo* script camera reference is synchronized. Other classes can subscribe to the *TransformGizmoManager OnTargetTransformChanged* event to be notified when the user transforms an instructions by using the gizmo.

3.3.5 Apprentice System Architecture

Figure 3.29 displays the controllers of the apprentice's application. The *ApprenticeController* is the apprentice's equivalent to the *MentorController* and, likewise, is the starting point of the application. It starts by presenting the start screen and subscribing to the *OnConnectToServer* event. When the user enters the IP address of the server and and a connection is established, the *ApprenticeController* switches the controller to the *ARApprenticeController*.

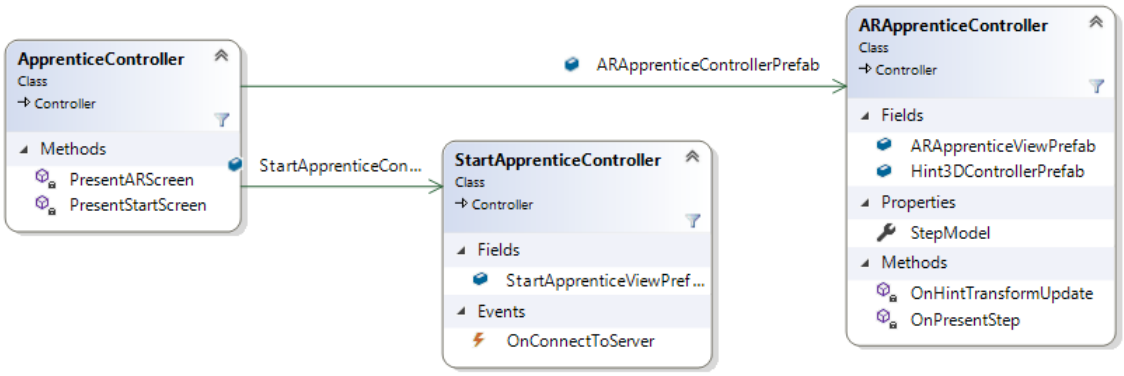


Figure 3.29: Class diagram displaying the apprentice application's controllers.

The *ARApprenticeController* subscribes to network events, using the *PresentStep* and *HintTransformUpdate* handles. When a *PresentStep* message arrives, the controller destroys the previous instructions 3D views and creates new views according to the *StepModel* deserialized from the message. When a *HintTransformUpdate* message arrives, the controller simply updates the corresponding hint's transform. This class is also responsible to send messages associated with the *StepCompleted* and *NeedMoreInstructions* network handles, when the user clicks the respective buttons.

Chapter 4

Evaluation

In this chapter, the experiments performed in order to test and validate the platform are described. These experiments are essential to answer the research questions of this project.

4.1 Experiment's Description

To test the platform, three types of experiments were prepared:

- labyrinth solving;
- painting of specific areas;
- LEGO pieces placement.

To answer the research questions of this project, a comparison between CooP AR and another communication tool is needed. Therefore, the task involving LEGO pieces was also performed using Skype[1], a typical video calls software. The labyrinth and paint experiments were excluded from the comparison with Skype because the information needed to accomplish these tasks would be very easy to be transmitted over Skype. In addition, the task involving LEGO pieces offered higher complexity and could more likely simulate problems in real-case scenarios.

To perform the experiments, groups of two persons were organized. In the first place, the project was explained to the participants. It was also informed that the participants could quit the experiment at any time and that the data collected was anonymized. Then, an apprentice and a mentor were chosen. The participants were separated and each one filled a form. The form asked about the participants' age group, if they had ever used an AR application before and how comfortable were they with using 3D software tools (e.g. Blender, 3Ds Max).

[1] Skype, an application used to communicate with other people over the internet, typically through video calls. URL: `https://www.skype.com` (visited on 06/11/2017).

Each group performed a total of three experiments: first the labyrinth, then the painting experiment and finally the experiment involving LEGO pieces. The mentor performed the tasks on a desktop computer, while the apprentice used an Android smartphone to run the client application. The experiments were performed on the same room, so that the responsible for the experiments could observe the participants simultaneously. However, the participants were not allowed to communicate with each other or see what each one was doing, as if they were on completely separated rooms.

4.1.1 Labyrinth Experiment

In this experiment, the apprentice had to solve a labyrinth printed on a sheet of paper using a pencil, following the virtual arrows that appeared on the screen of the smartphone (Figure 4.1). On the mentor's application, the steps were already defined, and the mentor had only to move to the next step when the apprentice notified that the step was completed. Each step was composed of three arrows and, after drawing the path on the paper for the given step, the apprentice would click on the *check mark* button. The users had to follow eight steps in total to complete the experiment, which corresponded to solve roughly half the labyrinth.

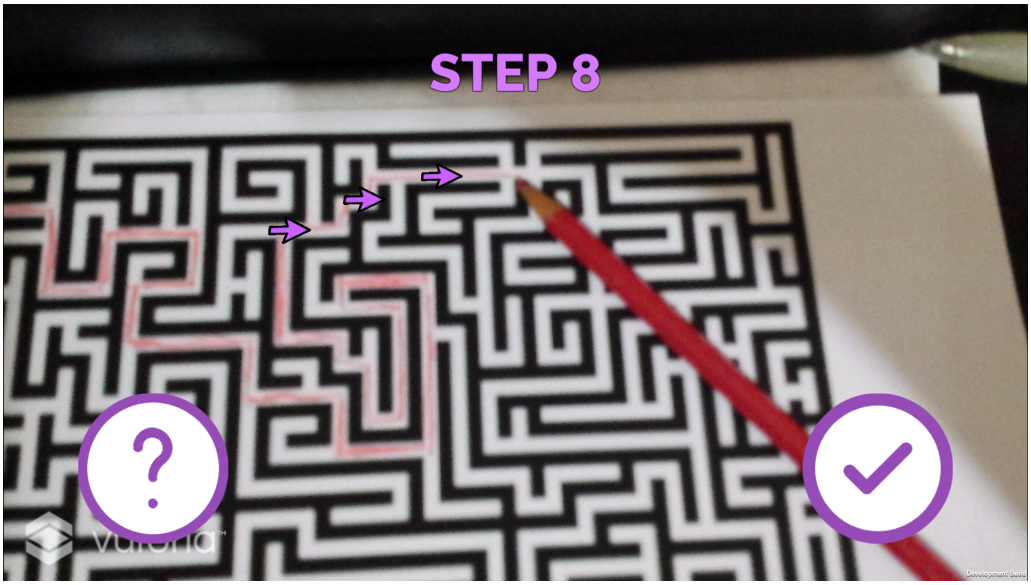


Figure 4.1: Labyrinth as viewed from the apprentice application.

For each group, this was the first of the experiments and, therefore, it serve as a way to introduce the platform to the users. In the beginning, it was explained to the apprentice and mentor how to use the user interface. For instance, it was explained that after completing a step, the apprentice would have to click on the *check mark* button to inform the mentor that the step was completed. On the other end, the mentor would have to check for notifications of the apprentice and advance for the next step.

After completing the task, the apprentice was required to fill a form about the experiment. The goal of this experiment was to test whether the augmented reality instructions were useful and if the communication between the users was processed without problems.

4.1.2 Paint Experiment

In this experiment, the apprentice had to paint certain areas of a drawing with a specific color, according to the instructions given. For instance, at the step 5 it would be required to paint with a green color an area indicated by an arrow (see Figure 4.2). The user had four different color pencils available.

Figure 4.2: Drawing as viewed from the apprentice application.

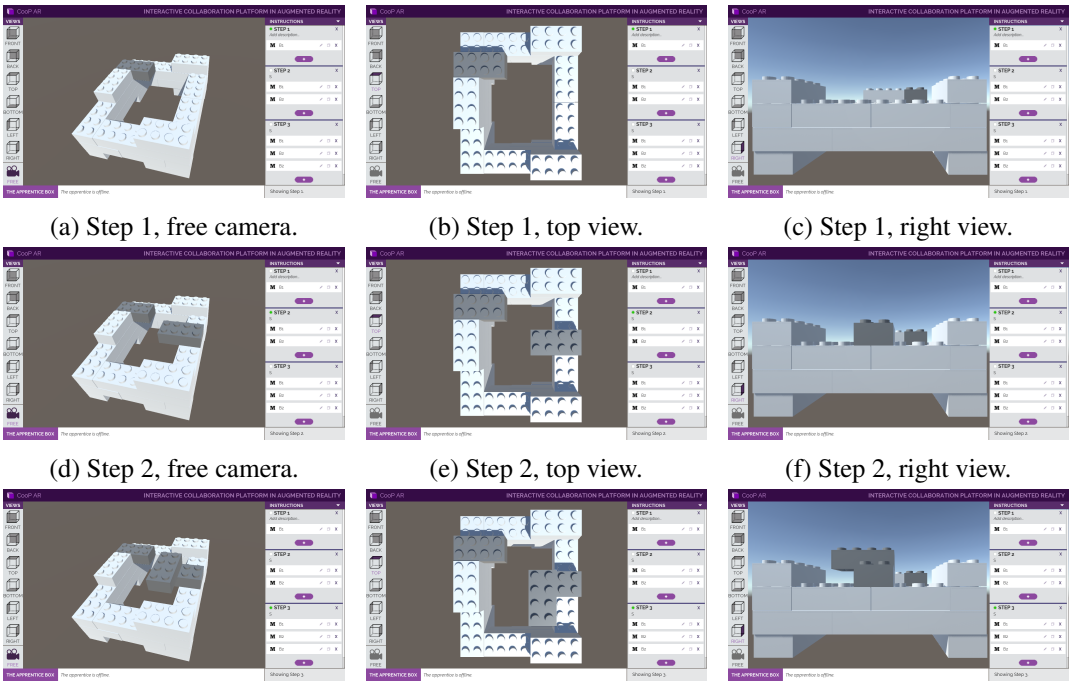
As in the previous experiment, the steps on the mentor application were already defined and the mentor only had to move to the next step when needed. At the end, the apprentice had to fill a form.

The goal of the experiment was to test whether the application could be used to signal and indicate very specific areas and if it was possible to transmit messages clearly. The apprentice had to follow seven steps in total, which targeted a range of large to very small and specific areas.

4.1.3 LEGO Experiment

In this experiment, the apprentice had to place three LEGO pieces onto specific spots. The mentor was given three sheets of paper (Figure 4.3) and the goal was to explain the apprentice on which places the pieces should fit.

One of the objectives of this experiment was to compare the collaboration platform with a typical software that allows video calls. The software chosen for comparison was Skype. Having this in consideration, the experiment had two versions: one using the collaboration platform and



(a) Step 1, free camera. (b) Step 1, top view. (c) Step 1, right view.

(d) Step 2, free camera. (e) Step 2, top view. (f) Step 2, right view.

(g) Step 3, free camera. (h) Step 3, top view. (i) Step 3, right view.

Figure 4.3: Images of the three sheets of paper given to mentor for the LEGO experiment. Each sheet of paper corresponded to a different step.

another using Skype. Each group performed only one of the versions. At the end of the experiment, both the apprentice and the mentor had to fill a form. Different questions were asked whether the participant was apprentice or mentor. The CooP AR and Skype forms had similar questions in order to have a term of comparison. However, the CooP AR form had additional questions that targeted specific aspects of the collaboration platform.

4.1.3.1 CooP AR

In this version of the LEGO experiment, the apprentice had to wait some minutes while the mentor prepared the task. The user interface commands were first explained to the mentor. Then, it was given some time so that the mentor could experiment the new commands (e.g. create a new geometry hint and use the transform gizmos).

Soon after, the task was explained to the mentor and the three sheets of paper (Figure 4.3) were given. In the first place, the mentor had to create a new hint representing a LEGO piece. Then, the LEGO piece had to be transform according to the images on the first sheet. In the second place, the mentor had to create a new step. By creating a new step, the previous hint was cloned, and the mentor had to clone it and transform it according to the second sheet. Finally, the mentor created the final step, cloned one of the hints and transformed the final piece. As the task was prepared, it was signaled that the collaboration session would begin and the apprentice connected to the system. After that, each experiment proceeded as the previous experiments (Figure 4.4).

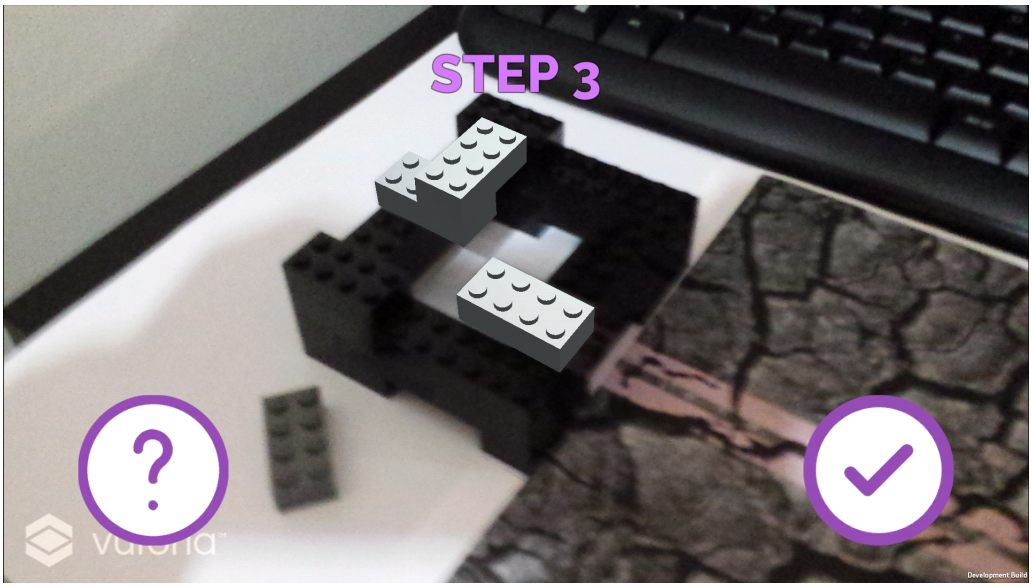


Figure 4.4: LEGO experiment view from the apprentice application.

One of the goals of this version of the experiment was to test the usability of the mentor's application, including the controlling of the cameras, the managing of the instructions and transformation of the 3D objects. Another goal was to test how good the system was on transmitting 3D information to the real world.

4.1.3.2 Skype

In this version, the apprentice and the mentor communicate through Skype in order to accomplish the task with the LEGO pieces. The mentor used a computer to perform the call, while the apprentice used a phone. The participants performed the experiment on the same room for two reasons: it was desired that the responsible for the experiment would be able to take notes while observing the participants simultaneously; and to eliminate the variable of the quality of the sound of the call. Thus, the call was performed with the speakers and microphones turned off and the participants could speak freely and hear each other perfectly. The restriction was that the mentor could only see what the apprentice was doing through the apprentice's phone camera.

4.2 Results and Analysis

In total, there were 30 participants, which were divided into groups of 2 persons, making up 15 groups. Each group performed 3 experiments: labyrinth, paint and LEGO. In addition, the groups performed the LEGO experiment either using CooP AR or Skype, in order to compare the performance of each tool.

The participants' ages varied between 17 and 55, with the majority being between 17 and 25 (Figure 4.5). From the 15 apprentices, only 7 had previously used an augmented reality application. From the 15 mentors, 6 had never used before a 3D software tool (e.g. Blender, 3ds Max).

The chart depicted on Figure 4.6 displays how comfortable were the mentor regarding using 3D software tools.

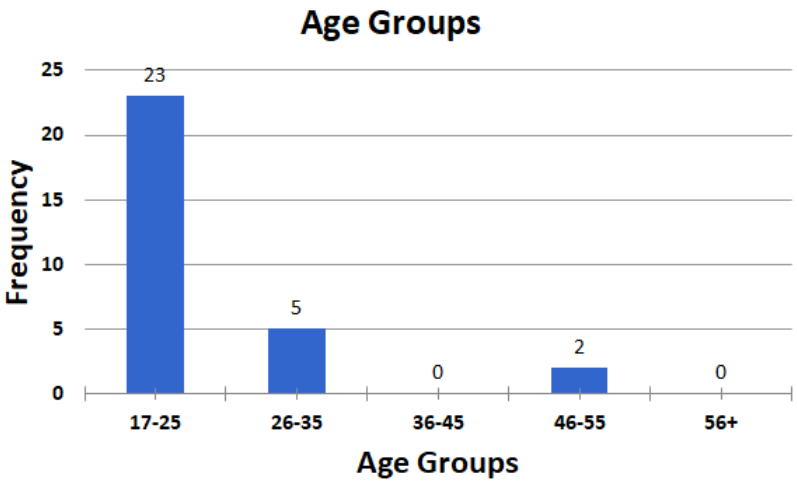


Figure 4.5: Ages of the participants.

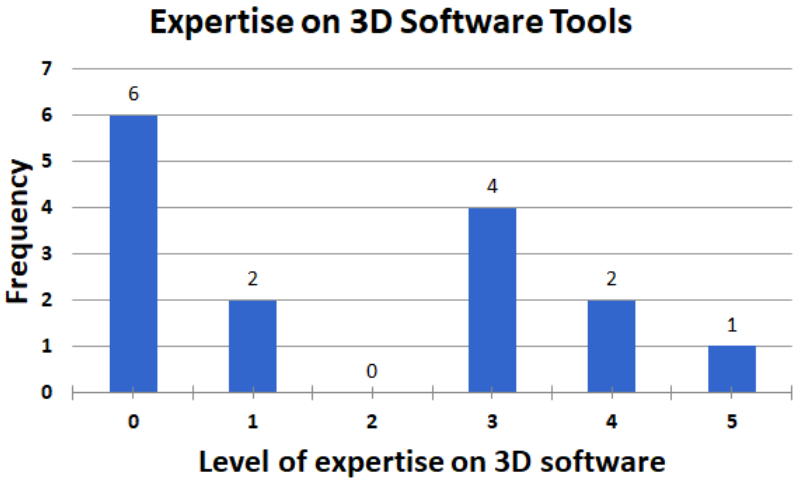


Figure 4.6: Level of expertise of the mentors on using 3D software tools. A value of 0 means that the participant never used a 3D software tool. First quartile: 0; median: 1; third quartile: 3.

4.2.1 Labyrinth Experiment

On every labyrinth experiment, the apprentice was able to follow the exact planned path. The initial steps took longer time, as users were still accustoming to the system: stabilizing the camera so that the visual tracking was not lost, having to hold the smartphone with one hand and writing with the other and understanding the whole process of the experiment. The completion of the following steps was generally much quicker.

In certain occasions, the apprentices took more time following the instructions, because they were always looking at the phone while tracing the path. As a single stroke of the pencil was not well visible by observing the smartphone screen, users spent more time retracing the lines to make sure they were visible. However, after some time they realized that they could just check the instructions at the screen and then look directly at the paper and trace the line, which was much more quicker.

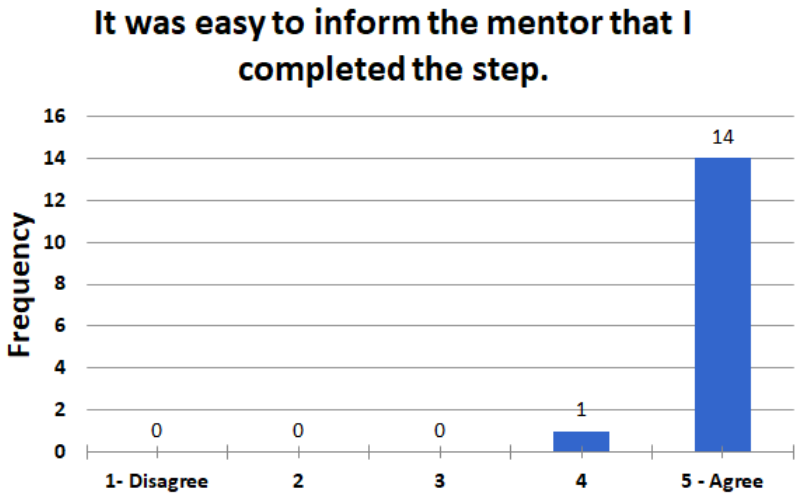


Figure 4.7: Answers to question on how easy it was to inform the mentor that a step was completed. First quartile: 5; median: 5; third quartile: 5.

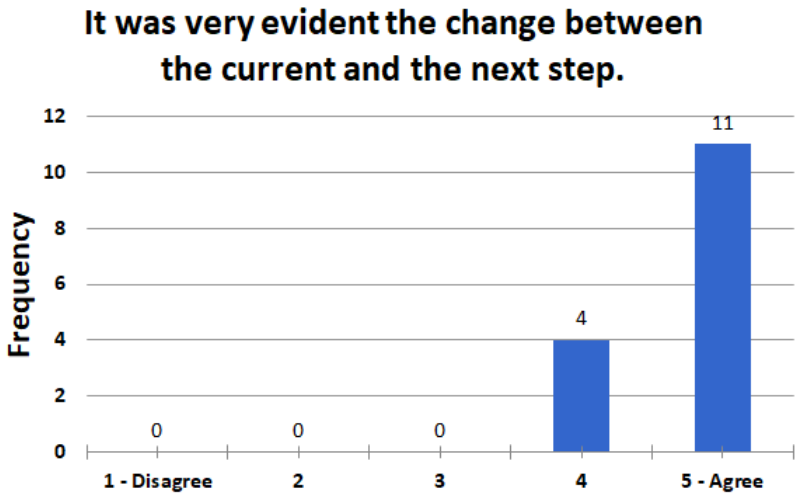


Figure 4.8: Answers to the question on how evident the changes between the steps were. First quartile: 4.5; median: 5; third quartile: 5.

The apprentices reported, through the forms, that it was very easy to inform the mentor that the task was completed (Figure 4.7). This was expected as users only had to click on a check-mark button in order to inform the mentor. In addition, users reported that it was well evident

the change between the steps (i.e. when the mentor switched to the following step) (Figure 4.8). When changing a step, a central label identifying the step number was updated as well as the virtual objects displayed through augmented reality.

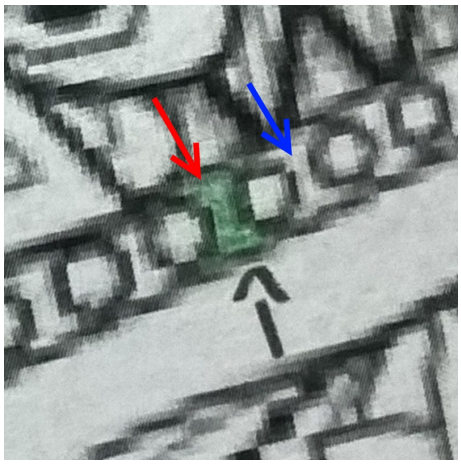
4.2.2 Paint Experiment

There were 15 experiments in total. As each task was composed of 7 steps, there were a total of $7 \times 15 = 105$ steps to complete. From these 105 steps, there was a total of 9 errors that were observed, which corresponds to an 8.6 % error rate, approximately (Equation 4.1). The average was 0.60 errors per experiment, with a standard deviation of 0.71. Table C.1, located in the Appendix C presents the number of errors per experiment in detail.

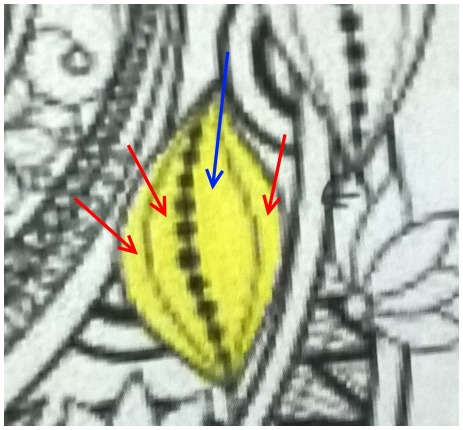
$$\frac{9}{7 \times 15} = \frac{9}{105} \approx 8.6\% \tag{4.1}$$

Several kinds of errors were observed. Only 1 of the errors was due to precision: the user painted on the wrong location (Figure 4.9a). 3 errors were due to the misinterpretation of the instructions. For instance, the user painted the whole leaf (Figure 4.9b), instead of just a part, or even several circles instead of just one (Figure 4.9c). In one of the experiments, one of the areas was unpainted which means that the step was unintentionally skipped. Finally, 4 of the errors observed were related with the step 7 of the task (Figure 4.9d). In this step, the target area was composed of a very slim margin that was difficult to see and 4 of the users managed to paint over it, targeting the whole leaf instead of just the inner part. There were a total of 8 experiments with no observed errors.

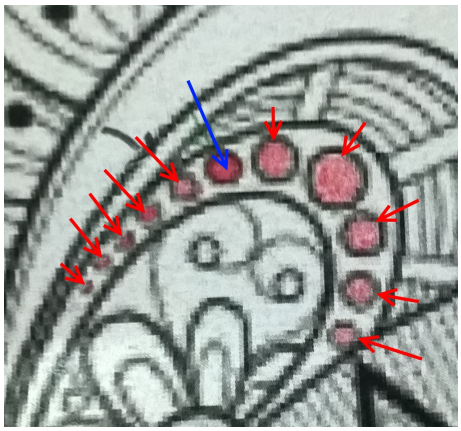
In this experiment, one particular observation was that sometimes the instructions were off-screen and the apprentices couldn't see it immediately. At some point they realized that they had to move the smartphone to reveal the instructions. In this case, as the space reserved to experiment was very restricted, this wasn't a problem. However, this emphasizes that visual cues that indicate points of interest off-screen are of great importance for more general and complex scenarios.



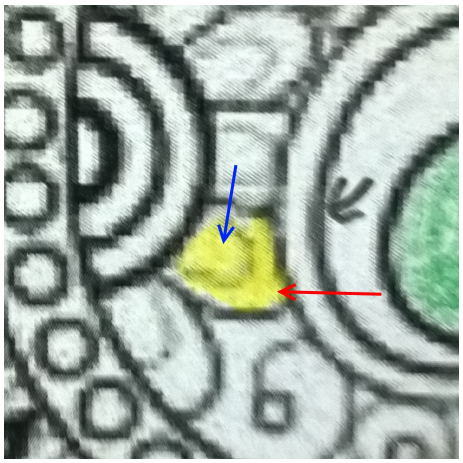
(a) User painted an area just beside the supposed one.



(b) User painted the whole leaf, instead of only the part indicated by the blue arrow.



(c) User painted all surrounding circles, instead of just the one indicated by the blue arrow.



(d) User painted the whole leaf, instead of just the inner part, possibly due to the margin delimiting it being too thin.

Figure 4.9: Errors performed by users on the paint experiment.

The apprentices reported, in general, that the instructions were very precise in space, as seen in Figure 4.10. This idea is enhanced by the fact that there was only 1 precision error. Globally, the users expressed that the instructions were clear (Figure 4.11). However, it can be concluded that the instructions should have been clearer, as 3 users reported and as several misinterpretation errors were committed.

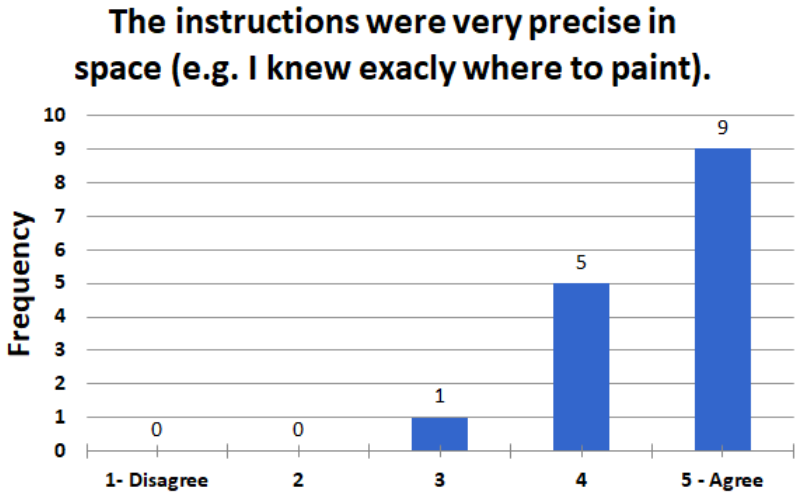


Figure 4.10: Answers to the question on how precise in space were the instructions. First quartile: 4; median: 5; third quartile: 5.

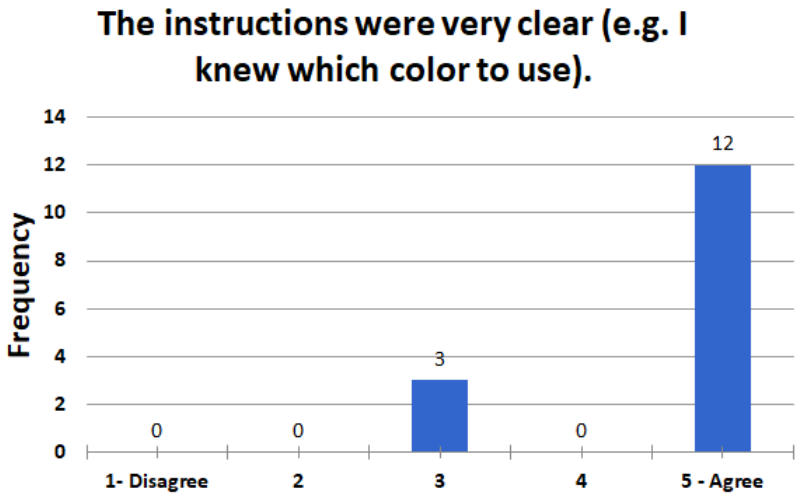


Figure 4.11: Answers to the question on how clear the instructions were. First quartile: 5; median: 5; third quartile: 5.

4.2.3 LEGO Experiment

For this experiment, there were a total of 15 experiments. 8 of them were performed using the collaboration platform, while the other 7 were accomplished using Skype.

4.2.3.1 Comparison between CooP AR and Skype

The users that used Skype employed very similar strategies. Before even beginning to place the pieces, some mentors asked the apprentice to orientate the base LEGO structure, so that it was oriented in the same manner as displayed on the given sheets of paper. This was a way to ensure

that mentor wouldn't be tricked by their perception of 3D space. Then, generally, for each piece the mentors started by giving an hint on where it should be placed. The apprentices would then try to place the piece where they understood it should be, which was almost always the wrong place. Then, the mentors would tell them to move the piece in a certain direction, or to rotate it 90 degrees. This same strategy was repeated over and over again until the three pieces were fit on the correct place.

Aside the common communication problems, such as bad video quality, one common problem on the video calls was that the mentor had not enough visibility on what the apprentice was doing. Therefore, many times the mentor asked to raise the smartphone in order to have a better perception of the LEGO pieces locations.

Mentors took several minutes, roughly between 5 and 15 minutes, on the preparation of the task itself using CooP AR. However, once the task was prepared and the apprentice connected, the instructions were so straightforward that the apprentices quickly placed the pieces in order, by just looking at the smartphone's screen.

Figure 4.12 displays a box plot of the durations of the experiments using CooP AR and Skype. In general, the CooP AR experiments took much less time than the Skype sessions (69 versus 172 seconds on average). One of the Skype experiments even took around 389 seconds.

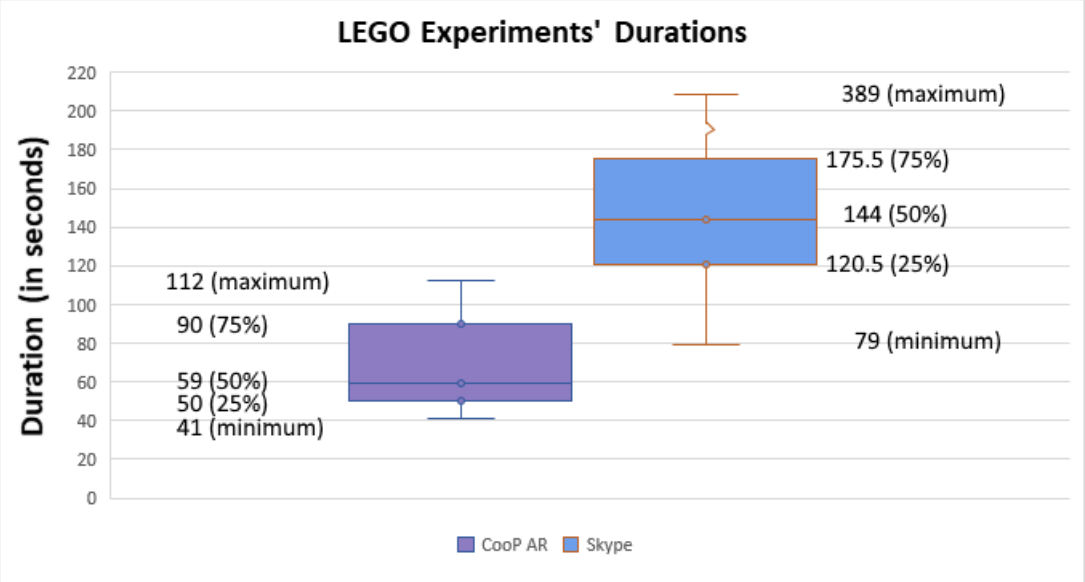


Figure 4.12: Box plot of the LEGO experiments durations, units in seconds. CooP AR mean: 69.13; CooP AR standard deviation: 24.50; Skype mean: 172.00; Skype standard deviation: 94.29.

Regarding the forms, users tended to give a better classification to CooP AR than Skype. Figure 4.13 displays the answers given to the question on if the respective software used made collaboration tasks easier. Users tended to thought that CooP AR made collaboration easier, possibly because the instructions were very straightforward and tasks were completed quickly. On the other hand, participants tended to give lower score to Skype, probably because instructions were hard to explain through Skype. This idea is enhanced by the results shown on Figure 4.14.

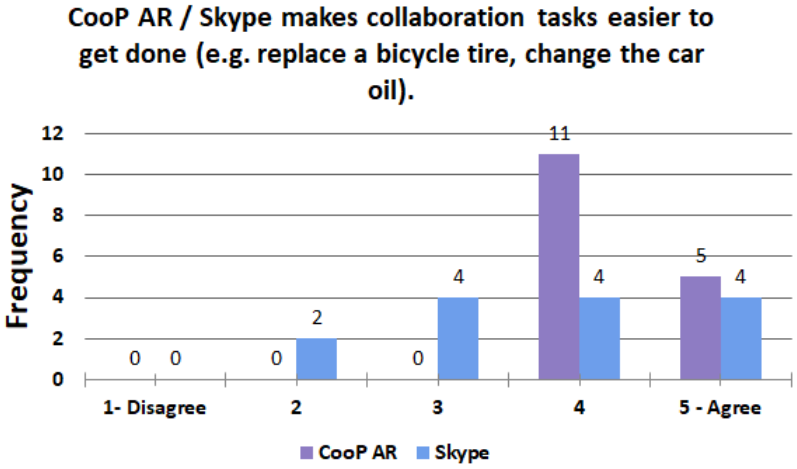


Figure 4.13: Answers to the question on if CooP AR / Skype make collaboration tasks easier. CooP AR first quartile: 4; CooP AR median: 4; CooP AR third quartile: 5; Skype first quartile: 3; Skype median: 4; Skype third quartile: 4.75.

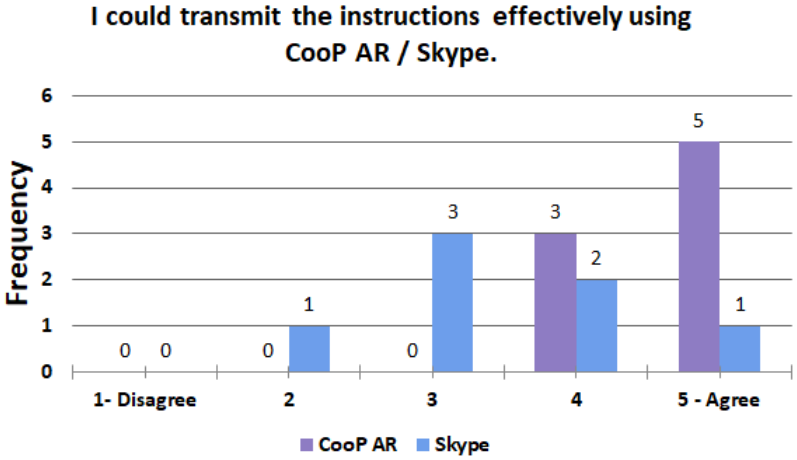


Figure 4.14: Answers to the question on if instructions are easy to transmit using CooP AR / Skype. CooP AR first quartile: 4; CooP AR median: 5; CooP AR third quartile: 5; Skype first quartile: 3; Skype median: 3; Skype third quartile: 4.

The form filled by the participants contained an open field for commenting positive and negative aspects of either CooP AR or Skype, depending on the version of the experiment. Participants praised CooP AR for being practical and easy to use.

One user mentioned that it was very useful that CooP AR could be deployed on a smartphone. However, the majority of negative comments were regarding the fact that it was difficult to perform the tasks while holding the smartphone and that using an AR device that provided an hands free experience would be ideal.

Regarding Skype, users complained that the quality of the video calls could interfere with performing the tasks, as the video stream often lagged. One user also mentioned that Skype would

possibly need much more network bandwidth than CooP AR.

4.2.3.2 CooP AR Usability

In this experiment, it was required that the mentor prepared the task itself. Therefore, the mentor had to create three steps. The last step would be composed by three geometry hints, each one corresponding to a LEGO piece. These LEGO pieces had to be placed according to the images printed on a sheet of paper.

The position and orientation of the pieces that originated theses images were annotated as the ideal values. Then, after each session of the LEGO task, the transformation values of each piece of the third step were saved into a file.

The original values and the saved ones were used to calculate a value of how precise each mentor placed each piece. Equation 4.2a calculates an offset value between the original and the saved positions. However, a certain value of $position_{offset}$ is difficult to interpret, because it requires a mean of comparison with a known measure. Therefore, the $position_{offset}$ was divided by 0.028, which is the length of a LEGO pin. Thus, Equation 4.2b gives the position offset in LEGO pins. Equation 4.2c calculates the difference between the orientations of an original and a saved piece.

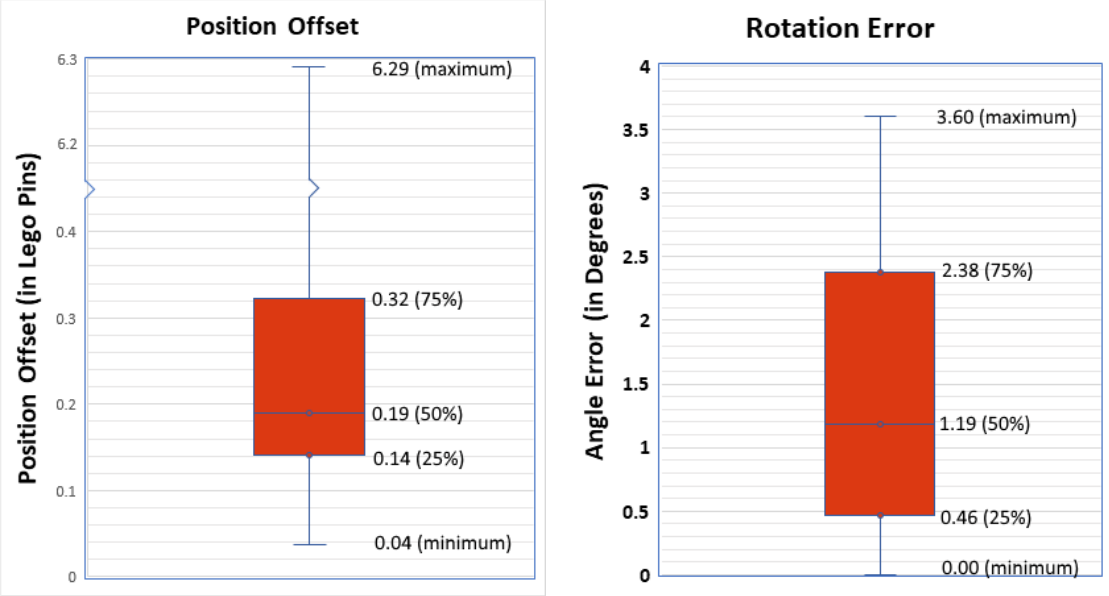
$$position_{offset}(original, saved) = \|original - saved\| \tag{4.2a}$$

$$position_{pinoffset}(original, saved) = \frac{position_{offset}(original, saved)}{0.028} \tag{4.2b}$$

$$rotation_{error}(original, saved) = \sum_{i=1}^{n} \mid original_i - saved_i \mid \tag{4.2c}$$

A higher value of $rotation_{error}$ corresponds to less precision on the orientation of the piece. For instance, if the values corresponded to 3D rotation vectors in degrees, a $rotation_{error}$ of 5.3 would mean that the piece is misaligned in 5.3 degrees.

Figures 4.15a and 4.15b display the box plots corresponding to the position offsets in LEGO pins units and rotation errors in degrees. Note that the position offsets were generally bellow $\frac{1}{3}$ of a LEGO pin, in spite of an outlier that reached 6.29 LEGO pins. From the collected data, we can infer that, in general, the mentors were able to place the pieces on the correct positions with reasonable precision, having into account that they had the images displayed on Figure 4.3 as guidelines. However, the presence of an outlier reveals that more experiments should be performed in order to increase the number of samples. The rotation errors varied between 0 and 3.6 degrees. In this particular experiment, from the point of view of the apprentice, the rotation errors were not significant as the pieces could only be fitted with 90 degrees of interval.



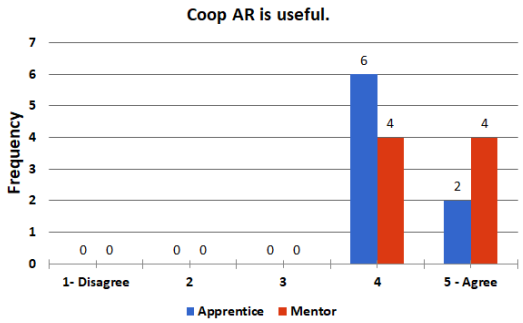
(a) Box plot of the positions' offsets. Mean: 0.46; standard deviation: 1.22.

(b) Box plot of the rotations' offsets. Mean: 1.48; standard deviation: 1.22.

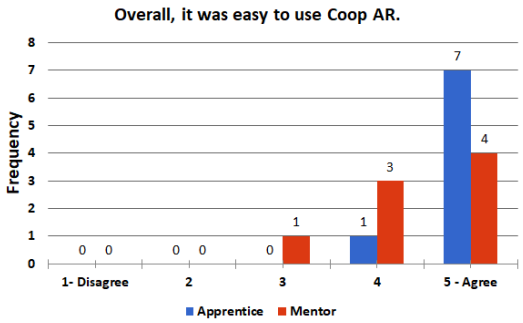
Figure 4.15: Precision results.

Figure 4.16 depicts the answers given by both mentors and apprentices to an usability form. Both group of users found out CooP AR useful. It can be noted that the apprentices agreed that the client application was very easy to use, whereas the mentors thought that theirs was more complex. This was expected, as the apprentice's application is composed by only two button, whereas the mentor's is much more complicated and includes several tools. In spite of being complex, the mentors generally agreed that they learned to use it quickly. In addition, both group of participants thought that the user interfaces were pleasant.

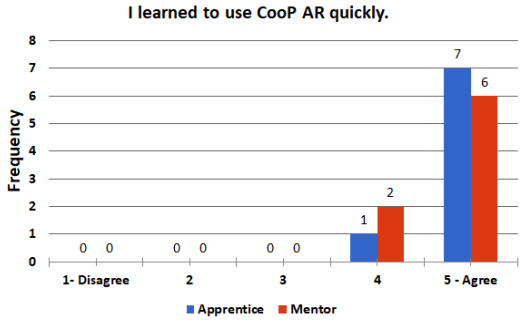
Figure 4.17 displays the answers of the mentors regarding the usability of specific tools within the application. Figures 4.17a and 4.17b demonstrate that some users had some difficulty changing between cameras. This could be due to the fact that the mentors were not used to the user interface and to control multiple points of view. In addition, even more mentors thought that the camera was not particularly easy to control. This is comprehensible, as, for instance, the free could be moved and rotated in multiple directions and it is difficult to grasp the controls at first. By analyzing Figure 4.17c, it's possible to conclude that some mentors didn't think the notifications regarding the apprentice's activity were not clearly seen. On the open comment field, one participant even mentioned that the notifications regarding the apprentice should be more noticeable. Figure 4.17d shows that users found it was rather easy to create new instructions. Regarding the transformation of objects, Figures 4.17e and 4.17f show that some users had some difficulties moving and, specially, rotating objects using the transform gizmo. This were also expected results and they confirm that the transform gizmo needs further improvements.



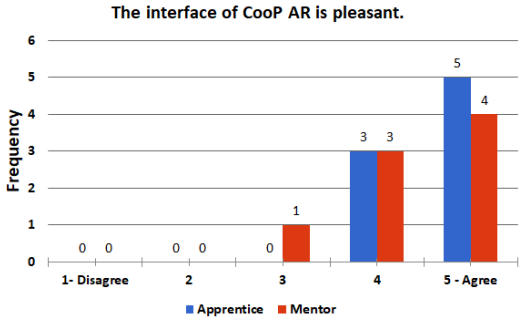
(a) Answers as to how useful CooP AR is. Apprentice first quartile: 4; Apprentice median: 4; apprentice third quartile: 4.25; mentor first quartile: 4; mentor median: 4.5; mentor third quartile: 5.



(b) Answers as to how easy CooP AR was to use.Apprentice first quartile: 5; Apprentice median: 5; apprentice third quartile: 5; mentor first quartile: 4; mentor median: 4.5; mentor third quartile: 5.

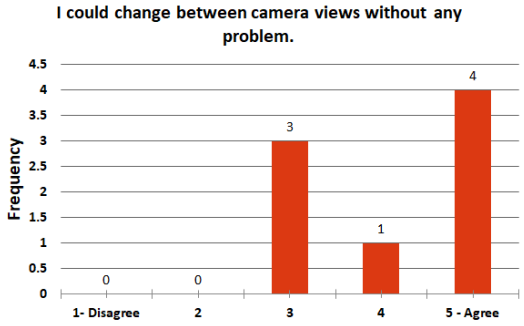


(c) Answers as to whether it was quick to learn how to use CooP AR. Apprentice first quartile: 5; Apprentice median: 5; apprentice third quartile: 5; mentor first quartile: 4.75; mentor median: 5; mentor third quartile: 5.

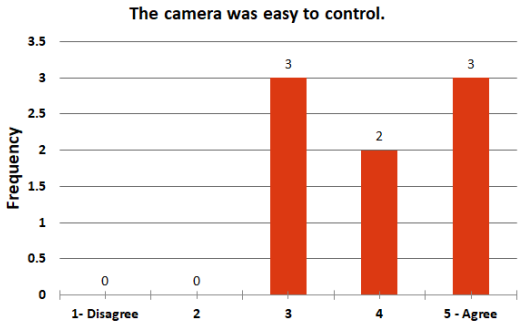


(d) Answers as to whether the CooP AR interface was pleasant. Apprentice first quartile: 4; Apprentice median: 5; apprentice third quartile: 5; mentor first quartile: 4; mentor median: 4.5; mentor third quartile: 5.

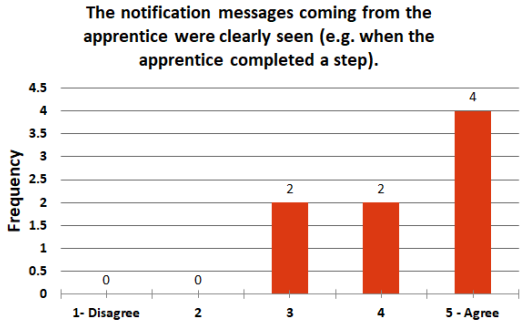
Figure 4.16: Usability form results.



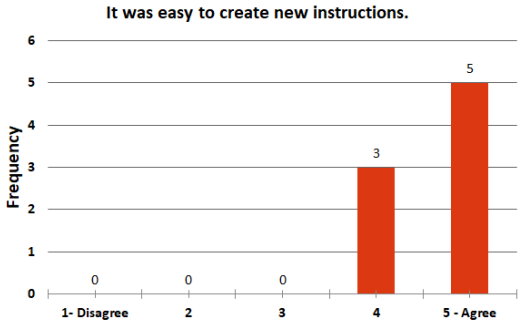
(a) Answers as to how easy was to change between different cameras. First quartile: 3; median: 4.5; third quartile: 5.



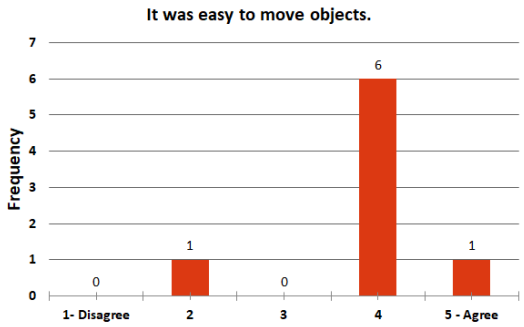
(b) Answers as to how easy was to control a camera. First quartile: 3; median: 4; third quartile: 5.



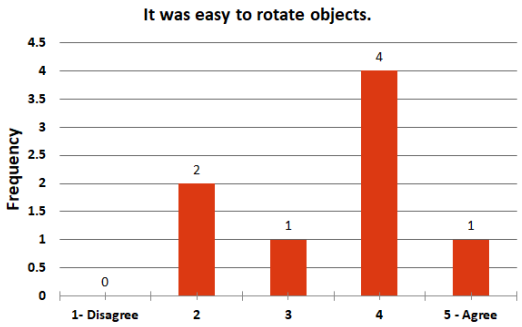
(c) Answers as to whether the notifications were clearly seen. First quartile: 3.75; median: 4.5; third quartile: 5.



(d) Answers as to how easy was to create new instructions. First quartile: 4; median: 5; third quartile: 5.



(e) Answers as to how easy was to move objects. First quartile: 4; median: 4; third quartile: 4.



(f) Answers as to how easy was to rotate objects. First quartile: 2.75; median: 4; third quartile: 4.

Figure 4.17: Usability of mentor tools form results.

4.2.4 Other Observations

One particular problem that did only rarely manifest during the experiments, but that could emerge much more often in real case scenarios is that the mentor couldn't validate the work of the apprentice. For instance, in the paint experiment, one of the apprentices missed a step and others painted areas that were not supposed to, and the mentor had no way to know that the apprentices failed to perform that step.

One possible solution could be to integrate a video stream from the apprentice to the mentor. This way, the mentor could validate each step, as the apprentice followed the instructions. Another solution, which would not increase the network bandwidth so much, would be to introduce a new feature in which, after each step, the apprentice takes a picture of the result, that is sent to the mentor for examination.

4.2.5 Discussion

In order to evaluate the collaboration platform, 30 participants divided in groups of 2 participated in three different experiments: labyrinth, paint and LEGO. The labyrinth and the paint experiments were performed using CooP AR, whereas the LEGO experiments were performed using either CooP AR or Skype, in order to study the impact of either tool on the experiment's task. For each group, the mentor and apprentice roles were assigned, and each experiment consisted in the mentor explaining the apprentice how a certain task would be solved, using the given experiment's tool.

In the labyrinth experiment, the mentor had to help the apprentice solve a labyrinth, using CooP AR. All apprentices were able follow the planned path, but some took more time than others due to several reasons: some were not accustomed to follow instructions using an AR application; whereas others were having difficulties tracing the path with a pencil with one hand, while having to hold the smartphone in place with the other. The goal of the experiment was to introduce the platform to the participants and to test the exchange of information. The apprentices reported that it was easy to inform the mentor that a step was completed and that it was clear when the mentor changed to the following step.

The paint experiment was more complex and required the apprentice to paint very specific areas of a figure to be colored. Some errors were observed, such as: drawing wrong areas due to precision and misinterpretation of the instructions, skipping steps and coloring more than intended due to a slim margin. These errors occurred in 8.6% of the steps.

For the LEGO experiment, the apprentice had to place three LEGO pieces in their correct places, according to the instructions given by the mentor. In some experiments, the users used CooP AR, whereas in others Skype was used. In the CooP AR experiments, the mentor was required to prepare the instructions using the CooP AR interface. In general, in this experiment's context, the precision with which the pieces were placed on the application was high. The mentor's reported that the application was easy to use, despite the opinion of some users that the objects were hard to rotate. When it came to perform the collaboration task, the groups that used CooP AR were able to complete the task, in average, 2.49 times quicker than the Skype's groups. Users that used Skype thought that it was not so easy to transmit the instructions and that it was propitious to communication fails.

Overall, the results display that CooP AR is useful for performing remote collaboration tasks and has some advantages regarding, for instance, video calls applications, while the users find it helpful and easy to use. However, the apprentices had some difficulty in performing the tasks with only one hand due to the use of the *smartphone*. Providing equipment, such as a tripod to hold the phone or head-worn displays, to allow an hands-free experience could help the execution of

the tasks and achieve better results. Although it did not have a significant influence on the results, it was observed on the paint experiment that some users could not see some instructions because they were off-screen. For larger scale environments, the existence of visual cues to help the user locate the instructions that may be off-screen is of great importance to the effective use of the platform. In addition, most of the errors that occurred on the paint experiment could be avoided if CooP AR had a validation mechanism implemented, in which the mentor could confirm the work performed by the apprentice, for instance, the by checking a photo sent by the apprentice's application. Despite this, the number of errors committed on the paint experiment was low, and, in the LEGO experiment, CooP AR performed better than the video calls.

Chapter 5

Conclusions

Augmented reality is useful in several contexts and it can be applied particularly to collaboration systems. It has the potential of enhancing teaching and augment collaboration by linking the virtual and real worlds. Also, AR user interfaces should be designed with care in order to provide a natural way of interaction with the system, which is the preferred way of interaction by users.

This dissertation proposes an augmented reality based framework to enhance remote collaboration. The framework includes a client-server architecture to support the roles of apprentice and mentor, which use the client and server applications respectively. The mentor uses the application in order to create instructions regarding a certain task. After the instructions are prepared, these are saved and can be used in any collaboration session, repeatedly. The apprentice can then connect to the system and visualize the instructions through augmented reality. The framework supports communication between the users, in order to allow the mentor to monitor the progress of the apprentice.

A software solution was developed in order to put the framework into practice, named CooP AR. The server application allows the mentor to create a task and divide it further into steps. Then, the mentor can describe each step using hints. The application lets the user to control several camera views and features a message box which displays information regarding the apprentice's activity. The hints can represent text, images or 3D geometry and can be transformed using a transform gizmo. While a task is selected, the apprentice's application can be used to display the instructions of the currently selected step on the mentor's application. The apprentice can inform that the step was completed by clicking a check-mark button, which instructs the mentor to move to the next step.

In order to test the platform, experiments were performed with a total of thirty participants. For each group of two users, three experiments were performed. The first experiment consisted on solving a labyrinth, the second in painting specific areas of a drawing and a the third to fit LEGO pieces in a certain manner. The two first experiences were performed using solely CooP AR, while the latter was performed either using CooP AR or Skype. The results showed that the

collaboration platform performed generally well, however, it has room to improve, particularly, regarding the usability of tools such as the transform gizmos. In addition, one of the goals of the experiment was to compare CooP AR with Skype. The collaboration platform had better results, in general, having users complete the required task in less time and with greater efficiency.

The objectives of this projects were, in general, accomplished.

5.1 Future work

CooP AR was developed considering the experiments; however, it still has several limitations that need to be overcome for more general usage.

Some of the limitations are related with changing certain values in the process of preparing a task, using the mentor application. For instance, it is not possible to change certain camera parameters, such as the projection matrix or the sensibility of zoom, using the application's user interface. Currently, this can only be changed using the Unity Editor. Another limitation is that the image targets used for visual tracking, as well as the static geometry, such as the LEGO base structure of the LEGO experiment, for a given task can only be changed using Unity.

Another limitation is that mentors can only use predefined geometry and images as hints. One solution could be to add a new option, when creating a new hint, to add a custom image/geometry from a file. The downside of this approach is that the custom image/geometry would have to be transferred through network to the apprentice's application.

Another problem which must be addressed in future work is that the developed system does not allow an effective communication between the apprentice and the mentor. The mentor should be able to validate the actions performed by the apprentice, possibly by sending photos at the end of each step or by integrating a video stream. In addition, the apprentice should be allowed to communicate with the mentor freely (e.g. to ask questions), preferably through voice.

The experiments performed in this project provided useful feedback in order to continuously improve the collaboration platform. As features are added and tweaked, new tests and experiments should be performed in a greater scale.

References

[ABB+01] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *IEEE computer graphics and applications*, 21(6):34–47, 2001.

[AGG+15] Clemens Arth, Raphael Grasset, Lukas Gruber, Tobias Langlotz, Ro Mulloni, Dieter Schmalstieg, and Daniel Wagner. The history of mobile augmented reality developments in mobile ar over the last almost 50 years, 2015.

[Ans01] Adnan Ansar. Registration for augmented reality, 2001.

[ARToolKit] Open source augmented reality sdk | artoolkit.org. URL: `https://archive.artoolkit.org/` (visited on 06/26/2017).

[Azu97] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[BFS04] Istvan Barakonyi, Tamer Fahmy, and Dieter Schmalstieg. Remote collaboration using augmented reality videoconferencing. In *Proceedings of Graphics interface 2004*, pages 89–96. Canadian Human-Computer Communications Society, 2004.

[BG96] Meera M Blattner and Ephraim P Glinert. Multimodal integration. *IEEE multimedia*, 3(4):14–24, 1996.

[BK02] Mark Billinghurst and Hirokazu Kato. Collaborative augmented reality. *Communications of the ACM*, 45(7):64–70, 2002.

[BK08] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[BKP08] Mark Billinghurst, Hirokazu Kato, and Ivan Poupyrev. Tangible augmented reality. *ACM SIGGRAPH ASIA*, 7, 2008.

[BTV06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[CFA+11] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, 51(1):341–377, 2011.

[CHD+11] Ceridwen Coulby, Scott Hennessey, Nancy Davies, and Richard Fuller. The use of mobile technology for work-based assessment: the student experience. *British Journal of Educational Technology*, 42(2):251–265, 2011.

[CLS+10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[Cos15] Eduardo Dias da Costa. Unity with mvc: how to level up your game development. Toptal. 2015. URL: `https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development` (visited on 06/25/2017).

[csevents17] C# reference (event). Microsoft. May 2017. URL: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event` (visited on 06/25/2017).

[gdec13] Understanding component-entity-systems. Boreal Games. April 2013. URL: `https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013` (visited on 06/25/2017).

[GHT11] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International journal of computer vision*, 94(3):335, 2011.

[GNT+14a] Steffen Gauglitz, Benjamin Nuernberger, Matthew Turk, and Tobias Höllerer. In touch with the remote world: remote collaboration with augmented reality drawings and virtual navigation. In *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology*, pages 197–205. ACM, 2014.

[GNT+14b] Steffen Gauglitz, Benjamin Nuernberger, Matthew Turk, and Tobias Höllerer. World-stabilized annotations and virtual scene navigation for remote collaboration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 449–459. ACM, 2014.

[GZB13] Mark Graham, Matthew Zook, and Andrew Boulton. Augmented reality in urban places: contested content and the duplicity of code. *Transactions of the Institute of British Geographers*, 38(3):464–479, 2013.

[Hai] Osian Haines. An introduction to simultaneous localisation and mapping | kudan, kudan blog article. URL: https://www.kudan.eu/kudan-news/an-introduction-to-slam/ (visited on 02/04/2017).

[HBH+17] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, pages 235–252. Springer, 2017.

[HF04] Tobias Höllerer and Steve Feiner. Mobile augmented reality. *Telegeoinformatics: Location-Based Computing and Services. Taylor and Francis Books Ltd., London, UK*, 21, 2004.

[HF09] Steven J Henderson and Steven Feiner. Evaluating the benefits of augmented reality for task localization in maintenance of an armored personnel carrier turret. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 135–144. IEEE, 2009.

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[JBC08] Carmen Juan, Francesca Beatrice, and Juan Cano. An augmented reality system for learning the interior of the human body. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, pages 186–188. IEEE, 2008.

[KBL15] Panos E Kourouthanassis, Costas Boletsis, and George Lekakos. Demystifying the design of mobile augmented reality applications. *Multimedia Tools and Applications*, 74(3):1045–1066, 2015.

[Kle06] Georg Klein. *Visual tracking for augmented reality*. PhD thesis, University of Cambridge, 2006.

[Kle96] Lindsay Kleeman. Understanding and applying kalman filtering. In *Proceedings of the Second Workshop on Perceptive Systems, Curtin University of Technology, Perth Western Australia (25-26 January 1996)*, 1996.

[KM07] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[KM13] Yu-Doo Kim and Il-Young Moon. E-training content delivery networking system for augmented reality car maintenance training application. *International Journal of Multimedia and Ubiquitous Engineering*, 8(2):69–80, 2013.

[KMW11] Nabeel Younus Khan, Brendan McCane, and Geoff Wyvill. Sift and surf performance evaluation against various image deformations on benchmark dataset. In *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*, pages 501–506. IEEE, 2011.

[Kudan] Home | kudan, kudan homepage. URL: `https://www.kudan.eu/` (visited on 01/28/2017).

[KudanScale16] Scale in simultaneous localisation and mapping | kudan, kudan blog article. URL: `https://www.kudan.eu/kudan-news/scale-simultaneous-localisation-mapping/` (visited on 02/03/2017).

[KudanTech16] Kudan slam engine. tech teaser, youtube teaser of the kudan's slam engine. URL: `https://www.youtube.com/watch?v=x-HlVprsalQ` (visited on 01/28/2017).

[LH08] Taehee Lee and Tobias Hollerer. Hybrid feature tracking and user interaction for markerless augmented reality. In *Virtual Reality Conference, 2008. VR'08. IEEE*, pages 145–152. IEEE, 2008.

[Lin15] Tony Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1):3–36, 2015.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[ML09] Xue Mei and Haibin Ling. Robust visual tracking using l1 minimization. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1436–1443. IEEE, 2009.

[MTU+95] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. In *Photonics for industrial applications*, pages 282–292. International Society for Optics and Photonics, 1995.

[mvp] The model-view-presenter (mvp) pattern. URL: `https://msdn.microsoft.com/en-us/library/ff649571.aspx` (visited on 06/25/2017).

[Nys17] Robert Nystrom. Observer · design patterns revisited · game programming patterns. May 2017. URL: `http://gameprogrammingpatterns.com/observer.html` (visited on 06/25/2017).

[OCW+00] S Oviatt, P Cohen, LZ Wu, J Vergo, L Duncan, B Suhm, J Bers, T Holzman, T Winograd, J Landay, J Larson, and D Ferro. Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions. *HUMAN-COMPUTER INTERACTION*, 15(4):263–322, 2000. ISSN: 0737-0024. DOI: 10.1207/S15327051HCI1504_1.

[OpenCV] Opencv | opencv, opencv homepage. URL: http://opencv.org/ (visited on 01/28/2017).

[PM06] Muriel Pressigout and Eric Marchand. Hybrid tracking algorithms for planar and non-planar structures subject to illumination changes. In *Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on*, pages 52–55. IEEE, 2006.

[PPS13] PM Panchal, SR Panchal, and SK Shah. A comparison of sift and surf. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):323–327, 2013.

[PYN98] Jun Park, Suya You, and Ulrich Neumann. Natural feature tracking for extendible robust augmented realities. In *Proc. Int. Workshop on Augmented Reality*, volume 2 of number 3, pages 2–2, 1998.

[QSS14] Murad Qasaimeh, Assim Sagahyroon, and Tamer Shanableh. A parallel hardware architecture for scale invariant feature transform (sift). In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pages 295–300. IEEE, 2014.

[RCR+14] Teresa Restivo, Fátima Chouzal, José Rodrigues, Paulo Menezes, and J Bernardino Lopes. Augmented reality to improve stem motivation. In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 803–806. IEEE, 2014.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[RGizmo] Unity3druntimetransformgizmo. URL: https://github.com/HiddenMonk/Unity3DRuntimeTransformGizmo (visited on 06/10/2017).

[RJA+16] E Gutiérrez de Ravé, FJ Jiménez-Hornero, AB Ariza-Villaverde, and J Taguas-Ruiz. Diedricar: a mobile augmented reality system designed for the ubiquitous descriptive geometry learning. *Multimedia Tools and Applications*:1–23, 2016.

[RLL+08] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1-3):125–141, 2008.

[RRK+11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[RSN07] DR Awang Rambli, S Sulaiman, and MY Nayan. A portable augmented reality lab. In *1st International Malaysian Educational Technology Convention (IMETC 2007)*, 2007.

[Skp] Skype, an application used to communicate with other people over the internet, typically through video calls. URL: `https://www.skype.com` (visited on 06/11/2017).

[SOB+05] Christian Sandor, Alex Olwal, Blaine Bell, and Steven Feiner. Immersive mixed-reality configuration of hybrid user interfaces. In *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, pages 110–113. IEEE, 2005.

[SSF+98] Zsolt Szalavári, Dieter Schmalstieg, Anton Fuhrmann, and Michael Gervautz. "studierstube": an environment for collaboration in augmented reality. *Virtual Reality*, 3(1):37–48, 1998.

[TDM+08] Johannes Tumler, Fabian Doil, Rudiger Mecke, Georg Paul, Michael Schenk, Eberhard A Pfister, Anke Huckauf, Irina Bockelmann, and Anja Roggentin. Mobile augmented reality in industrial applications: approaches for solution of user-related issues. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 87–90. IEEE Computer Society, 2008.

[Tur14] Matthew Turk. Multimodal interaction: a review. *Pattern Recognition Letters*, 36:189–195, 2014.

[unetcon17] Unity - manual: network system concepts. Unity Technologies. 2017. URL: `https://docs.unity3d.com/Manual/UNetConcepts.html` (visited on 06/25/2017).

[Unity] Unity - game engine, unity homepage. URL: `https://unity3d.com/` (visited on 01/28/2017).

[Unreal Engine] Unreal engine technology | home, unreal engine homepage. URL: `https://www.unrealengine.com` (visited on 01/28/2017).

[Unreal4AR, Unreal4AR homepage] Unreal4ar. URL: `http://www.unreal4ar.com/` (visited on 01/28/2017).

[VLL+14] Jean-Luc Vinot, Catherine Letondal, Rémi Lesbordes, Stéphane Chatty, Stéphane Conversy, and Christophe Hurter. Tangible augmented reality for air traffic control. *interactions*, 21(4):54–57, 2014.

[VP07] D Van Krevelen and R Poelman. Augmented reality: technologies, applications, and limitations, 2007.

[Vuforia] Vuforia | augmented reality, vuforia homepage. URL: https://www.vuforia.com/ (visited on 01/28/2017).

[WBE+11] Sabine Webel, Uli Bockholt, Timo Engelke, Matteo Peveri, Manuel Olbrich, and Carsten Preusche. Augmented reality training for assembly and maintenance skills. In *BIO Web of Conferences*, volume 1, page 00097. EDP Sciences, 2011.

[WS07] Daniel Wagner and Dieter Schmalstieg. *Artoolkitplus for pose tracking on mobile devices*. na, 2007.

[WS09] Daniel Wagner and Dieter Schmalstieg. History and future of tracking for mobile phone augmented reality. In *International Symposium on Ubiquitous Virtual Reality*, pages 7–10. IEEE, 2009.

[ZBW+14] Zhiwei Zhu, Vlad Branzoi, Michael Wolverton, Glen Murray, Nicholas Vitovitch, Louise Yarnall, Girish Acharya, Supun Samarasekera, and Rakesh Kumar. Ar-mentor: augmented reality based mentoring system. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 17–22. IEEE, 2014.

[ZD03] Saso Zagoranski and Saša Divjak. *Use of augmented reality in education*, volume 2. IEEE, 2003.

[ZDB08] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: a review of ten years of ismar. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE Computer Society, 2008.

REFERENCES

Appendix A

Forms

A.1 Participants

Questions:

- 1.1. User ID.
- 1.2. Group ID.
- 2. Age group.
- 3.1. Have you ever used a 3D software tool (e.g. AutoCAD, Blender, Maya, 3DStudio, Unity)?
- 3.2. If yes, how comfortable are you with using a 3D software tool?
- 4. Have you ever used an augmented reality application?

Table A.1: Participants

1.1	1.2	2	3.1	3.2	4
0	0	46-55	Yes	3	No
1	0	46-55	No		No
2	2	26-35	Yes	4	Yes
3	2	26-35	Yes	3	No
5	3	17-25	No		Yes
4	3	17-25	Yes	4	Yes
6	4	26-35	Yes	4	Yes
7	4	26-35	No		Yes
9	5	17-25	No		Yes
8	5	17-25	No		Yes
11	6	17-25	Yes	4	Yes
10	6	17-25	Yes	2	Yes
12	7	17-25	Yes	3	Yes
13	7	17-25	Yes	5	Yes
15	8	17-25	No		Yes
14	8	17-25	Yes	1	No
17	9	17-25	Yes	1	Yes
16	9	17-25	Yes	2	No
19	10	17-25	Yes	3	Yes
18	10	17-25	Yes	3	No
20	11	17-25	Yes	1	No
21	11	17-25	Yes	3	Yes
22	12	17-25	Yes	3	No
23	12	26-35	No		Yes
24	13	17-25	No		Yes
25	13	17-25	Yes	1	Yes
27	14	17-25	Yes	4	Yes
26	14	17-25	Yes	2	No
28	15	17-25	Yes	2	No
29	15	17-25	No		Yes

A.2 Labyrinth

Questions:

- 1. User ID.
- 2.1. Augmented reality instructions were useful for completing the given tasks.

- 2.2. It was easy to inform the mentor that I completed the step.
- 2.3. It was very evident the change between the current and the next step.

Table A.2: Labyrinth

1	2.1	2.2	2.3
1	4	5	5
2	5	5	5
4	5	5	5
6	5	5	5
8	5	5	4
10	5	5	5
12	4	5	5
14	3	5	4
16	5	5	5
18	5	5	5
20	4	4	4
22	4	5	5
24	5	5	5
26	5	5	4
28	5	5	5

A.3 Paint

Questions:

- 1. User ID.
- 2.1. The instructions were very precise in space (e.g. I knew exactly where to paint).
- 2.2. The instructions were very clear (e.g. I knew which color to use).
- 3.1. Did you find the need to ask for more instructions, in case they weren't clear enough?
- 3.2. It was easy to ask for help, in order to clarify the instructions.
- 3.3. The new instructions given by the mentor were helpful.

Table A.3: Paint

1	2.1	2.2	3.1	3.2	3.3
1	5	5	No		
2	5	3	No		
4	4	5	No		
6	5	3	No		
8	4	5	No		
10	5	3	Yes	4	4
12	5	5	No		
14	4	5	No		
16	5	5	No		
18	4	5	No		
20	3	5	No		
22	5	5	No		
24	4	5	No		
26	5	5	No		
28	5	5	No		

A.4 LEGO

A.4.1 CooP AR

Common Questions:

- 1. User ID.
- 2.1. Which was your role? (implicit on tables)
- 2.2. CooP AR makes collaboration tasks easier to get done (e.g. replace a bicycle tire, change the car oil).
- 3.1. Coop AR is useful.
- 3.2. CooP AR does everything I would expect it to do.
- 3.3. Overall, it was easy to use Coop AR.
- 3.4. I could use CooP AR without written instructions.
- 3.5. I didn't notice any inconsistencies as I used CooP AR.
- 3.6. I learned to use CooP AR quickly.
- 3.7. The interface of CooP AR is pleasant.

- 4. If you'd like, mention any positive and negative points of CooP AR.

Apprentice specific questions:

- A.1. When using CooP AR, I was able to complete the given tasks.
- A.2. I believe I became quickly productive using Coop AR.
- A.3. CooP AR was pleasant to use.

Table A.4: CooP AR LEGO Apprentices

1	2.2	3.1	3.2	3.3	3.4	3.5	3.6	3.7	A.1	A.2	A.3
1	4	4	4	5	5	5	5	5	5	4	5
2	4	4	5	5	3	3	5	4	4	5	4
8	4	4	5	5	4	4	5	5	5	4	5
10	4	4	5	5	4	4	4	4	5	5	5
14	5	4	4	4	5	5	5	5	5	4	5
22	5	4	5	5	3	5	5	5	5	3	5
24	4	5	4	5	5	5	5	5	5	4	5
26	5	5	3	5	5	5	5	4	5	5	4

Mentor specific questions:

- M.1. I could transmit the instructions effectively using Coop AR.
- M.2. I could change between camera views without any problem.
- M.3. The camera was easy to control.
- M.4. The notification messages coming from the apprentice were clearly seen (e.g. when the apprentice completed a step).
- M.5. It was easy to create new steps.
- M.6. It was easy to create new instructions.
- M.7. It was easy to move objects.
- M.8. It was easy to rotate objects.

Table A.5: CooP AR LEGO Mentors

1	2.2	3.1	3.2	3.3	3.4	3.5	3.6	3.7	M.1	M.2	M.3	M.4	M.5	M.6	M.7	M.8
0	4	5	4	4	4	4	5	5	5	4	4	5	4	4	4	3
3	5	4	4	4	3	2	4	3	4	3	3	3	5	5	4	2
9	4	4	4	5	5	5	5	5	5	5	4	5	5	5	5	5
11	4	5	4	5	5	5	5	4	5	3	3	4	4	4	2	2
15	4	4	4	4	3	5	5	4	4	5	5	5	4	4	4	4
23	5	5	5	5	5	5	4	5	5	5	5	5	5	5	4	4
25	4	5	5	5	5	3	5	5	5	5	5	3	5	5	4	4
27	4	4	4	3	5	4	5	4	4	3	3	4	5	5	4	4

Answers to question 4:

- User ID 2: "Positive - It's ease to use / Negative - For the experience developed is better use a AR glasses"
- User ID 8: "Pratical and usefull"
- User ID 14: "Having to hold the phone leaves only 1 free hand. VR Glasses would probably work better."
- User ID 15: "Very simple and uncomplicated"
- User ID 26: "Positive: portability. Having the application in your phone is definitely a bonus. Negative: portability, in itself, through the smarphone, makes me only able to use one hand, making my task more difficult."
- User ID 27: "It would be better if the notification that the apprentice did something had an animation or was more noticeable."

A.4.2 Skype

Questions:

1.1. User ID 1.2. How long did the experience take? 2.1. Which was your role? 2.2. Skype makes collaboration tasks easier to get done (e.g. replace a bicycle tire, change the car oil). A.1. When using Skype, I was able to complete the given tasks. A.2. I believe I became quickly productive using Skype. M.1. I could transmit the instructions effectively using Skype.

- 1. User ID.
- 2.1. Which was your role? (implicit on tables)
- 2.2. Skype makes collaboration tasks easier to get done (e.g. replace a bicycle tire, change the car oil).

- 4. If you'd like, mention any positive and negative points of Skype.

Apprentice specific questions:

- A.1. When using Skype, I was able to complete the given tasks.
- A.2. I believe I became quickly productive using Skype.

Table A.6: Skype Apprentice

1.1	1.2	2.1	2.2
4	3	4	2
6	5	5	5
12	4	5	4
16	5	5	5
18	4	5	5
20	3	4	3
28	4	4	4

Mentor specific questions:

- M.1. I could transmit the instructions effectively using Skype.

Table A.7: Skype Mentor

1.1	1.2	2.1
5	5	5
7	4	3
13	3	4
17	5	4
19	2	2
21	3	3
29	2	3

Answers to question 4:

- User ID 7: "Sujeito a falhas na chamada. Latencia do streaming de video."
- User ID 12: "Connection quality sometimes suffer"
- User ID 16: "The video call syncing was very off"
- User ID 17: "Higher bandwidth needs than the CooP AR, possibly"
- User ID 18: "Pode acontecer algumas falhas de imagem/som que pode prejudicar a finalização das tarefas."

Appendix B

Execution Times

Table B.1: Labyrinth execution times

Group ID	Execution Time
0	0:02:22
2	0:01:21
3	0:00:56
4	0:03:04
5	0:01:06
6	0:01:02
7	0:00:55
8	0:01:15
9	0:01:43
10	0:01:31
11	0:01:59
12	0:00:59
13	0:01:06
14	0:02:04
15	0:01:18

Table B.2: Paint execution times

Group ID	Execution Time
0	0:03:20
2	0:01:47
3	0:02:37
4	0:03:19
5	0:01:33
6	0:01:50
7	0:01:31
8	0:01:35
9	0:01:42
10	0:03:19
11	0:03:19
12	0:01:42
13	0:01:34
14	0:02:44
15	0:01:36

Table B.3: LEGO experiment, CooP AR execution times

Group ID	Execution Time
0	0:01:52
2	0:01:36
5	0:00:47
6	0:00:51
8	0:00:41
12	0:01:05
13	0:01:28
14	0:00:53

Table B.4: LEGO experiment, Skype execution times

Group ID	Execution Time
3	0:02:58
4	0:02:24
7	0:01:47
9	0:01:19
10	0:02:53
12	0:06:29
15	0:02:14

Appendix C

Errors and Precision Metrics

Table C.1: Paint experiment errors

Group ID	Precision Errors	Misinterpretation Errors	Skip Errors	Slim Margin Errors
0	0	0	0	0
2	0	0	0	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	0
6	0	1	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	1
13	0	0	0	1
14	0	1	0	1
15	1	0	0	0

Table C.2: First LEGO piece metrics

Group ID	Position			Rotation		
Reference	0.538	0.101	0.082	0.000	90.000	0.000
0	0.534	0.101	0.083	0.000	89.236	0.000
2	0.540	0.103	0.082	0.000	89.531	0.000
5	0.542	0.102	0.083	0.000	90.081	0.000
6	0.539	0.105	0.081	0.000	92.972	0.000
8	0.549	0.101	0.084	0.000	90.195	1.897
12	0.534	0.101	0.083	1.317	89.361	0.427
13	0.539	0.102	0.081	-0.888	90.764	1.944
14	0.529	0.102	0.084	0.000	90.502	0.000

Table C.3: Second LEGO piece metrics

Group ID	Position			Rotation		
Reference	0.707	0.101	0.004	0.000	90.000	0.000
0	0.713	0.101	0.000	0.000	89.236	0.000
2	0.708	0.103	0.001	0.000	89.531	0.000
5	0.707	0.102	0.003	0.000	90.081	0.000
6	0.550	0.103	0.084	0.000	87.720	0.000
8	0.708	0.101	-0.002	0.000	90.195	1.897
12	0.706	0.101	0.002	1.317	89.361	0.427
13	0.705	0.102	-0.001	-0.888	90.764	1.944
14	0.704	0.102	0.002	0.000	90.502	0.000

Table C.4: Third LEGO piece metrics

Group ID	Position			Rotation		
Reference	0.676	0.134	-0.024	0.000	0.000	0.000
0	0.684	0.135	-0.030	0.000	-0.448	0.000
2	0.679	0.137	-0.027	-0.333	0.776	-0.503
5	0.676	0.136	-0.024	0.000	0.110	0.000
6	0.680	0.140	-0.031	0.000	0.000	0.000
8	0.684	0.137	-0.029	0.000	0.059	1.897
12	0.681	0.133	-0.027	0.690	1.144	1.766
13	0.673	0.138	-0.038	-0.618	-0.026	-1.857
14	0.672	0.139	-0.027	0.000	-0.224	0.000